Least Authority
PRIVACY MATTERS

QuipuSwap Smart Contracts
Security Audit Report

Tezos Foundation

Final Report Version: 9 March 2021

Table of Contents

Overview

Background

Least Authority performed a security audit of the QuipuSwap Smart Contracts for the Tezos Foundation. QuipuSwap is intended to provide an easy and efficient way to exchange tokens and XTZ on the Tezos blockchain in a wide variety of ways. The QuipuSwap Smart Contracts aim to allow users to add their tokens to exchange, invest liquidity, and potentially make a profit in a fully decentralized way. The current implementation supports both the FA1.2 and FA2 standards.

Project Dates

- **November 19 - December 18:** Code review *(Completed)*
- **December 22:** Delivery of Initial Audit Report *(Completed)*
- **February 1 - 10:** Phase 1 Verification *(Completed)*
- **February 12:** Delivery of Updated Audit Report *(Completed)*
- **March 5 - 8**: Phase 2 Verification *(Completed)*
- **March 9**: Delivery of FInal Audit Report *(Completed)*

Review Team

- Phoebe Jenkins, Security Researcher and Engineer
- Sajith Sasidharan, Security Researcher and Engineer
- Nathan Ginnever, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the QuipuSwap Smart Contracts 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:

- QuipuSwap: https://github.com/madfish-solutions/quipuswap-core

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

```
3c1ce6f63081059ccf7155cb9574a348aef43f78
```

For the initial verification and follow up review, we examined the Git revision:

```
f4daf9389b98eac4fb2d4326ce0647d4f4e0a6a4
```

For the final verification, we examined the Git revision:

```
c0bd6b43bc9b3d98fb6887d946c999287c52555f
```

This repository was cloned for use during the audit and is linked for reference in this report:

https://github.com/LeastAuthority/quipuswap-core

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:

- README: https://github.com/madfish-solutions/quipuswap-core/blob/master/README.md
- FA1.2 Standard (TZIP-7): https://gitlab.com/tzip/tzip/-/blob/master/proposals/tzip-7/tzip-7.md
- FA2 Standard (TZIP-12): https://gitlab.com/tzip/tzip/-/blob/master/proposals/tzip-12/tzip-12.md
- QuipuSwap Demo: https://quipuswap.com/
- QuipuSwap Documentation: https://docs.quipuswap.com/

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

We commend the QuipuSwap development team for their achievements in implementing an ambitious system design. Additionally, we found that QuipuSwap utilized the Ligo language in interesting and innovative ways. Similar to other complex smart contract systems, QuipuSwap warrants further and ongoing analysis for potential vulnerabilities as applications in Decentralized Finance (DeFi) are rapidly evolving.

Review Scope

Our team's review of QuipuSwap covered the smart contracts system design and implementation, including configuration processes, deployment of the contracts, potential attacks resulting from the possibility for improper configuration and exploitation of the usage of lambdas, and the implementation of the standardized FA1.2 and FA2 interfaces. In addition, we examined common attack vectors against this functionality and verified the post-deployment operation of the Decentralized Exchange (DEX) contracts, such as the use of executing stored code.

Although the scope of the review was sufficient, there are a few areas that have been identified throughout the report for further review and analysis. These include the use of floating point operations, the JavaScript dependencies, and the voting and veto functionality, along with a more broad exploration of the potential interactions between contracts, such as modelling them as finite state machines.

System Design

Multi-stage Deployment

Our team has observed the QuipuSwap system design to be particularly complex in a number of ways. The system uses a multi-stage deployment process: it deploys the Factory contract, configures it by adding the desired functions, then uses it to deploy an additional DEX contract. This initial setup consists of different phases and relies on a number of steps being executed in a particular order, as opposed to issuing a single origination command for a typical deployment process. This process also increases the complexity of the implementation in several places, as it adds an additional level of abstraction to the code base (i.e. the contracts that are generated by the contracts found in the code base are being run as opposed to running the contracts themselves).

However, it is not clear if this multi-stage deployment process can be avoided, given the restrictions placed by Tezos on the amount of storage that can be used in a single transaction. Tezos contracts have hard-coded storage limits, which prove too low for the size of QuipuSwap contracts. Breaking up the process into multiple steps helps to ensure that each individual step stays below the storage limit. We recommend that the QuipuSwap development team explore future opportunities to simplify this approach if it is possible to do so.

Preprocessor Macros

Another source of increased complexity is the use of preprocessor macros, which configure functionality in individual components. While this design decision is warranted in that it helps avoid redundant and duplicated code throughout the system, the particular ways in which these macros are integrated throughout the code base cause the same lines of code to have different meanings in different contexts. This leads to increased difficulty in reviewing or auditing the code, as a result, we suggest documenting all variations and the reasoning behind the design choices of those variations into a specification (Suggestion 1).

Token Voting Governance

The QuipuSwap Smart Contracts also use a governance system to control the baker. To our knowledge, token voting governance has not fully matured and may present incentivization challenges to effectively change state when needed. There may be a desire to extend token voting governance to other economic settings in pair contracts, which will also require careful incentive research. We encourage decentralized governance as an appealing solution to centralized power and vulnerability, which result from single points of failure. We recommend that the voting and veto functionality undergo more thorough and focused analysis.

Floating Point Operations

The floating point operations used in the QuipuSwap contracts might potentially lead to edge cases. They use arbitrary precision numbers, rounding operations, and divisions -- all of which are known to lack precision when done by computers. As a result, we recommend further manual analysis, testing, and verification of these computations.

Complex Contract Interactions

Due to the complexity of the system, more broadly exploring the potential interactions between the contracts by modelling them as finite state machines and using some form of invariant analysis would provide valuable insight into the overall security of the system.

Code Quality

We found that the code is well-organized and compartmentalized in a clean and logical way. The code has good test coverage for the intended scenarios and common failure cases. However, we recommend expanding test coverage to include all error cases. This allows for tests that purposefully fail when contracts execute them in order to determine if error conditions operate and catch problems as expected (Suggestion 4). We also suggest including additional test coverage for the functions in `MetadataStorage` (Suggestion 5).

Furthermore, tests in the QuipuSwap code base are currently run with `npm`. We recommend running tests with `yarn`, as it is expected to result in a less error-prone testing process (Suggestion 6). Finally, adding Continuous Integration (CI) to the development process would ensure tests are always run and help detect certain errors at an earlier stage of development (Suggestion 7). More extensive coverage of error cases allow developers and reviewers to understand both intended and unintended scenarios, thus reducing the risk of code errors that lead to security vulnerabilities.

Documentation

Due to the complexity of the system architecture, gaining a concise overview and deep understanding of the way in which individual functions work is not a simple, intuitive process. As a result, we suggest integrating additional code comments and documentation, as detailed below.

Some limited comments are present throughout the code base, however, additional and more extensive code comments would be beneficial, particularly around the intent and usage of entry points and helpers (Suggestion 2). We found that error messages in the code report numeric codes, which can lead to a lack of clarity to what the error in question is and, as a result, makes troubleshooting the systems more challenging. We suggest clarifying the error message to include the causes of failure (Suggestion 3).

The existing project documentation is accurate and helpful. Nevertheless, given that certain design trade offs are required, resulting in increased system complexity, we recommend thoroughly documenting the overall system design and architecture, which would make the system more accessible to reviewers, contributors, and users of the code (Suggestion 11). In addition, better documentation of the interactions with other DeFi contracts would make the potential attack surfaces of such interactions more visible (Suggestion 9). Conducting and publishing more research on potential front-running attacks relating to Tezos would contribute towards mitigation against such attacks, even though it would not completely prevent their occurrence (Suggestion 10). As noted above, we recommend creating documentation that provides more detailed information on the intended behavior, functionality, states, and actions of the contracts, in order to facilitate a better understanding of the code by both security reviewers and contributors (Suggestion 1). Improvements to the documentation help to minimize potential human error as it relates to execution of the implementation, in addition to full comprehension by security reviewers of the code.

Dependencies

The QuipuSwap contracts are independent and self-contained such that they do not rely on external dependencies. However, we have identified several dependency concerns relating to the supporting JavaScript code used for testing and deploying the contracts. JavaScript package dependencies could be a security concern in this context, as there is no guarantee that all of them have been audited and maintained according to security best practices. We recommend that the JavaScript dependencies be further investigated and, where possible, well-known and previously audited dependencies should be used.

Additional Findings & Known Issues

Following the completion of the delivery of the *Initial Audit Report,* we conducted the Phase 1 Verification in which we identified one new issue and made one new suggestion.

We found that in the event that a shareholder does not interact with the contract for a period greater than thirty days, they would miss the opportunity for a reward, as the reward would not be updated with the correct frequency. As a result, we suggest tracking the number of reward cycles that have passed, so that calculating the missing reward is feasible and the opportunity for rewards can be retained (Issue A).

In addition, the QuipuSwap development team made significant improvements to the documentation. We commend this effort and recommend including a proof-reading stage to the development of documentation, in order to eliminate grammatical errors, which would help ensure that information is being conveyed adequately, with clear and unambiguous definition (Suggestion 12).

Finally, upon request from the QuipuSwap team, we verified the resolutions to several known issues (*Known Issues A - E*) that were identified by the QuipuSwap team.

Specific Issues, Suggestions, & Known Issues

We list the 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.

ISSUES, SUGGESTIONS, & KNOWN ISSUES	STATUS
Issue A: Shareholders Lose Opportunity for Rewards if Not Interacting with Contract	Resolved
Suggestion 1: Document the Specifications of Contracts	Partially Resolved
Suggestion 2: Expand Code Comments	Resolved
Suggestion 3: Better Define Error Messages	Resolved
Suggestion 4: Add Tests for Error Cases	Resolved
Suggestion 5: Add Test Coverage for MetadataStorage	Resolved
Suggestion 6: Implement Testing with yarn	Resolved
Suggestion 7: Add Continuous Integration to Development Process	Resolved
Suggestion 8: Consider using Formal Verification	Unresolved
Suggestion 9: Document Complex Contract Interactions	Resolved
Suggestion 10: Document Front-Running Attacks As They May Relate To Tezos	Resolved
Suggestion 11: Document Overall System Architecture	Resolved
Suggestion 12: Improve Documentation Readability	Unresolved

Known Issue A: Divestments With Zero XTZ Causes Tezos Error	Resolved
Known Issue B: XTZ to Token Ratio Can Be Manipulated During…	Resolved
Known Issue C: Parameters to Liquidity Divestment Call are Ignored	Resolved
Known Issue D: Users Able To Empty Token Reserves In Transfer…	Resolved
Known Issue E: High Initial Liquidity Causes Prohibitively Expensive Shares	Resolved

Issues

Issue A: Shareholders Lose Opportunity for Rewards if Not Interacting with Contract

Location
contracts/partials/MethodDex.ligo#L99

Synopsis
Every thirty days, a user is intended to receive a proper share of additional XTZ received from the delegation of the XTZ pool. This reward is provided to the user upon interaction with the contract, rather than calculated all at once. This is done to prevent a large gas cost being added to the operation for updating the reward information for all users. However, this means that a user would miss out on the reward if they do not interact with the contract for a period greater than thirty days, as the reward would not be updated with the correct frequency.

Preconditions
A user must be owed an award, but have not claimed that award via any actions with the contract that would trigger `update_user_reward`. Then, another reward cycle must pass, such that usually a user would be marked to receive another reward, but here the state would be unchanged due to the user already being marked.

Remediation
A possible solution would be to keep track of the number of reward cycles that have passed. If a user is several cycles behind, this will be reflected in their number of shares being unchanged and would allow for calculating the missing reward.

Status
The QuipuSwap team has responded and noted that, due to the operation size limit in Tezos, they have simplified the rewards model in order to decrease complexity and mitigate baker rewards distribution issues inside the protocol. In implementing the new rewards system, they collect all the baker rewards and simple XTZ transfers on the contracts for 30 days (a single cycle) and then distribute them in the next 30 days, during which the contract collects the rewards for distribution in the next iteration. Once users receive share tokens, they begin to receive rewards proportionally to their shares in the pool. We have verified that the mechanism of dispersal has changed and is no longer subject to this issue.

Verification
Resolved.

Suggestions

Suggestion 1: Document the Specifications of Contracts

Synopsis

The descriptions of major entry points contained in the README file can be considered an informal specification. However, expanding the documentation to provide more detailed information about actions that the contracts are expected to perform, intended behavior and functionality, and any illegal and/or unauthorized inputs and states they should be handling would facilitate a better understanding of the code.

Mitigation

We recommend documenting a specification of the QuipuSwap system, particularly specifying the calculations for and invariants in the following actions:

- Invest liquidity;
- Divest liquidity;
- Conversions between tokens;
- Rewards;
- Withdrawals of profits;
- DEX functions;
- Token functions;
- Vote; and
- Veto.

Status

A significant amount of documentation has been added, providing helpful information on the available functions and the possible circumstances which may result in failure.

It would additionally useful to document both the FA2 and the FA1.2 contracts, in order to easily assess the implementation's adherence to the respective specifications. This is of particular importance for the FA2 contracts since the FA2 standard (TZIP-12) has not yet been finalized, thus it is critical to understand what version of the specification is in use for the implementation.

Verification

Partially Resolved.

Suggestion 2: Expand Code Comments

Location

update_owner and update_metadata in contracts/main/MetadataStorage.ligo

middle_dex, middle_token, and use_default in contracts/partials/Dex.ligo

wrap_transfer_trx, set_dex_function, and set_token_function in contracts/partials/Factory.ligo

get_token_metadata_registry and update_operators in contracts/partials/MethodFA12.ligo

Synopsis
Further clarifications on the intent and usage of the main entry points and helpers functions would assist in reasoning about these functions and would limit the potential for misinterpreting their intended function.

Mitigation
We recommend adding or expanding code comments for the main entry points and helpers functions.

Status
Comments have been added to provide background information and context for much of the functions. In larger functions, such as `launch_exchange` in Factory.ligo, intermediate comments that have been added to allow for an easy understanding of the intended behavior.

Verification
Resolved.

Suggestion 3: Better Define Error Messages

Location
Uses of `failwith(“01”)`and similar in:

contracts/partials/Factory.ligo

contracts/partials/MethodFA12.ligo

Synopsis
The current error messages resulting from failures report numeric codes, which do not make explicitly clear the reason for the error in question. Instead, a system using transparent error messages would facilitate easier understanding and troubleshooting of the failure.

Mitigation
We recommend implementing more transparent and informative error messages, such as explicit error strings or descriptions of the cause of the failure.

Status
The error messages have been updated and more clearly defined, providing the user with information such as where the failure occurred and the reasoning for its occurrence.

Verification
Resolved.

Suggestion 4: Add Tests for Error Cases

Location
/test

Synopsis
For each use case, tests are currently run for a single successful scenario and check for expected values when the system is used as intended, but are absent for expected failures.

For example, the test for swapping tokens does a single swap and checks that the expected tokens are moved in the correct way. This could be expanded to encompass several scenarios where failure is expected, such as attempting to swap tokens that do not exist for the pair, trying to swap Tezos in a token only swap, or trying to swap amounts that are not aligned with the provided liquidity.

Finally, comprehensive test coverage helps to identify simple errors and prevents functionality from breaking when new code changes are introduced into the implementation, which minimizes the potential for errors leading to security vulnerabilities.

Mitigation

Create tests that will purposefully fail when the contract executes them, in order to explore if error conditions operate and catch problems as expected.

Status

The test suite has been updated, including a number of expected failure cases along with correct functioning under intended usage.

Verification

Resolved.

Suggestion 5: Add Test Coverage for `MetadataStorage`

Location

/test

Synopsis

The test suite does not exercise functions implemented in contracts/main/MetadataStorage.ligo.

Mitigation

Expand test coverage to exercise functions in contracts/main/MetadataStorage.ligo .

Status

Tests have been added to cover the `MetadataStorage` contract, including both successful expected behavior and failure cases.

Verification

Resolved.

Suggestion 6: Implement Testing with `yarn`

Synopsis

Running tests via `npm run test` has been an error-prone process. Our team made several attempts to run `npm run test` in a variety of environments, including local and cloud virtual machines, different operating systems, and Docker. In each of these instances, npm failed to install certain dependencies necessary in order to run the test suite. The QuipuSwap development team suggested using `yarn test` (present in pull request 11 of QuipuSwap's GitHub project) as an alternative to `npm run test`, which has worked as a more suitable alternative.

Mitigation
Switch to `yarn` for running tests.

Status
The test suite has been updated and now runs using `yarn`.

Verification
Resolved.

Suggestion 7: Add Continuous Integration to Development Process

Synopsis
The QuipuSwap project on GitHub lacks CI to run tests on new commits in the project. Implementing CI helps to prevent certain classes of issues early on in the development process.

Mitigation
Use CI to automatically run tests on new commits to the GitHub repository.

Status
QuipuSwap now utilizes Github Actions for continuous integration for all contracts.

Verification
Resolved.

Suggestion 8: Consider using Formal Verification

Location
contracts/partial/Factory.ligo

Synopsis
Since QuipuSwap performs computations that involve arithmetic rounding and arbitrary precision numbers, formal verification of those specific computations may help prevent a subclass of correctness issues. We recognize that formal verification will not be able to detect and prevent all classes of security and correctness issues, as focusing only on safety properties of specific areas of execution may not be able to fully define complex state machines or their interaction with other contracts. Logic issues may still arise even with extensive formal verification as illustrated by the recent Aave vulnerability.

Mitigation
We recommend that formal verification be considered for QuipuSwap. Projects similar to QuipuSwap, such as Uniswap and Dexter, have been formally verified. At this time, we do not have a particular recommendation for a specific formal verification tool or approach, however, we recommend that various approaches be further investigated.

Status
The QuipuSwap development team is investigating formal verification for Quipuswap, and will reassess the possibility of utilizing it at a later stage in the project. They have stated that they agree it is an important step for ensuring the validity and correctness of contracts on the Tezos blockchain.

Verification
Unresolved.

Suggestion 9: Document Complex Contract Interactions

Synopsis
Explicit documentation on how the QuipuSwap exchange may interact with other DeFi contracts would make the potential attack surfaces of such interactions visible.

For example, if a contract relies on the QuipuSwap exchange rates for a price feed as an oracle, they may be attacked by price manipulations if price checkpoints and the ability to time weight the price feed is not present. This attack could be facilitated with a flash loan.

Mitigation
We suggest documenting the capabilities of other projects to use the QuipuSwap platform and the mitigation methods that it provides, if applicable, against known DeFi oracle price manipulation attacks.

Status
The documentation has been updated so that it provides some insights into the limitations of QuipuSwap, alongside its features and expected interactions with the larger DeFi ecosystem. The documentation includes additional information on the various participants that make up a QuipuSwap exchange and their interactions.

Verification
Resolved.

Suggestion 10: Document Front-Running Attacks As They May Relate To Tezos

Synopsis
Issues associated with bots attempting to order transactions in sandwich attacks (front-running and back-running) and other front-running attacks have been prevalent in the Ethereum DeFi ecosystem. Front-running is difficult to address in Ethereum, or arguably any open ledger, due to anonymity/pseudo-anonymity, the predictable nature of transaction ordering in the client, public nature of the memory pool of transactions, and the inability to obfuscate the contract code.

Mitigation
This is an interesting problem that may have different consequences on the Tezos ledger than the Ethereum ledger. We suggest doing more research in the area of front-running attacks on the Tezos ledger and referencing the latest known research on the Ethereum ledger, specifically related to sandwich attacks. While this research does not provide a solution to sandwich attacks that are free from user experience tradeoffs, documenting the challenge on the QuipuSwap DEX as it relates to the sandwich attack would be a first step towards mitigation.

Status
The QuipuSwap development team provided existing papers investigating the possibility of front-running attacks and an analysis of the QuipuSwap governance system. Though the existence of these documents resolves this issue, we recommend including them in the QuipuSwap documentation to make them more easily accessible.

Verification
Resolved.

Suggestion 11: Document Overall System Architecture

Synopsis
In addition to the existing documentation on the API endpoints, documenting the high-level functionality of the system as a whole and how it is expected to be used would help explain some under-documented functionality (e.g.what problems voting is intended to solve) to end users, security reviewers, and potential developers.

Mitigation
We recommend accompanying the architectural diagram with a less-technical introduction to the project and its goals, as well as a high-level description of the different functionality and how it is expected to work within the system.

Status
The QuipuSwap online documentation has been updated to provide additional information on the system architecture, including architecture diagrams, high-level entity interaction diagrams, and QuipuSwap's smart contract API.

Verification
Resolved.

Suggestion 12: Improve Documentation Readability

Location
https://docs.quipuswap.com/

Synopsis
The documentation and code comments have a number of grammatical errors, which may result in confusion about the intended meaning. For example, the Prices and Fees section has a lot of helpful information, but the process of understanding it is slowed due to assumptions of knowledge and terminology, and grammatical errors adding further ambiguity.

Mitigation
Adding a proof-reading stage to the development of documentation, optimally by a technical reader who is not working on the QuipuSwap project, would help ensure that information is being conveyed adequately, with clear and unambiguous definition.

Status
Improvements to the documentation readability have not been made at the time of this verification. We encourage the QuipuSwap team to adopt documentation best practices, as suggested, in order to improve readability and understanding of their system.

Verification
Unresolved.

Known Issues

Known Issue A: Divestments With Zero XTZ Causes Tezos Error

Location

contracts/partials/MethodDex.ligo#L569

Synopsis

If a user calls `divest_liquidity` with a small number of shares, such that the amount of XTZ returned would be zero, Tezos generates an error message due to an empty transaction.

Remediation

Prevent divestments from amounting to zero XTZ, which will prevent this condition from being raised and avoid triggering the undefined behavior.

Status

Users have been prevented from withdrawing if the divested XTZ is equal to zero.

Verification

Resolved.

Known Issue B: XTZ to Token Ratio Can Be Manipulated During Operation

Location

contracts/partials/MethodDex.ligo#L569

Synopsis

When the user divests liquidity, the price of the token will almost always need to be truncated down, which will cause the ratio, or the invariant, to change over time. This can cause the token to effectively increase in value. There could also be the opposite effect such that, when divesting liquidity, the amount of shares is so small that the amount of tokens withdrawn would be zero. Compounding many operations of withdrawing XTZ without withdrawing tokens could result in dramatic changes in ratio.

Remediation

Preventing the withdrawal of tokens when the number of tokens divested would be zero prevents the situation where the ratio change could be dramatic, with many compounding operations.

Since floating point operations will always be imperfect, changes of value during divestment are unavoidable. For example, whether withdrawing 2.1 and 2.9 tokens, it will be rounded down to 2.0 tokens. The value of XTZ lost in this operation could be a large difference depending on the ratio. While this could be accepted as standard operation of the contract, a check could be put in place to ensure that this difference does not exceed a specific threshold.

As the invariant is updated after operations that modify the XTZ in reserve, this should not cause a desynchronization with the value of the token during contract operation. In general, we do not believe that this value should skew too much in any one direction. Many operations are present that modify the ratio in various directions and there will be financial incentives that correct this over time.

Status

A check has been added to prevent the withdrawal of XTZ without tokens. Currently, there are no checks in place regarding the divergence implicit with divesting liquidity, however, we do not consider this to present negative security implications.

Verification

Resolved.

Known Issue C: Parameters to Liquidity Divestment Call are Ignored

Location

contracts/partials/MethodDex.ligo#L569

Synopsis

Calls to `divest_liquidity` accept `min_tez` and `min_tokens` parameters, but no check is performed to ensure the divested amount confirms to the user's expectation.

Remediation

Add checks to ensure that the XTZ and tokens being divested are equal to or greater than the specified values.

Status

The `divest_liquidity` function now includes a check to make sure the divested amounts confirm to the user's expectation.

Verification

Resolved.

Known Issue D: Users Able To Empty Token Reserves In Transfer

Location

contracts/partials/MethodDex.ligo#L439

Synopsis

When users initiate a transfer that supplies XTZ in exchange for tokens, it is possible to entirely empty the contract's token reserves. This has the side effect of locking the contract, since many operations will be broken if the token balance is 0, thus causing an infinite price ratio.

Remediation

Prevent users from being able to transfer out the entire token supply by limiting the amount of the token pool able to be purchased at any given time. Additionally, add a check that prevents the token pool from having the ability to reach zero.

Status

A limit has been added to prevent users from transferring out more than one third of the token pool in a single transaction. In addition, this prevents the pool from ever reaching zero. Due to the way Tezos handles division, attempting to transfer out tokens when the value is 2 or less would be blocked for being greater than a third of the pool.

Verification

Resolved.

Known Issue E: High Initial Liquidity Causes Prohibitively Expensive Shares

Location

contracts/partials/Factory.ligo#L25

Synopsis

Since the amount of initial shares is hardcoded to be 1000, it is possible to provide a significant amount of initial liquidity, causing the cost per share to be prohibitive to most participants. This results in the entire exchange being controlled by a small number of users.

Mitigation

Have a variable or configurable amount of initial shares for the contract. Setting the number of shares equal to the initially provided XTZ (in mutez) prevents the cost of an individual share from being too expensive. Depending on intended behavior, it could also potentially be valuable to set a minimum number of initial shares, to ensure that the creator of the exchange starts with a competitive number of shares, but this does not represent a security issue.

Status

The initial shares are now variable, based on the amount of XTZ provided during exchange creation.

Verification

Resolved.

Recommendations

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

We commend the QuipuSwap development team on the improvement of both the project documentation and test coverage, which contributes to the overall quality of the system design and implementation. Documentation now includes the system architecture, the expected behavior of the contracts, complex contract interactions, and the possibility for front-running attacks.

However, we suggest that the QuipuSwap development team include a proof-reading stage to the development of documentation, in order to avoid the potential for misunderstanding of intended behavior or functionality. We also recommend that they continue to explore adding formal verifications to the project, which may prevent several types of correctness issues.

FInally, we commend the team for expanding tests to cover error cases, implementing tests with `yarn` instead of `npm`, and running tests on all new commits to the GitHub repository.

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.