Custody Free Quipuswap Wrapper for Kolibri DAO

Tezos Foundation

Independent security assessment report

inference

Report version: 1.1 / date: 28.11.2022

Table of contents

Version / Date	Description
1.0 / 14.11.2022	Final version
1.1 / 28.11.2022	Version 1.1, which fixes a functionality bug in the code. There was no security issue based on this bug.

Summary

Inference AG was engaged by the Tezos Foundation to perform an independent security assessment of the Custody Free Quipuswap Wrapper for the Kolibri DAO.

Inference AG performed the security assessment based on the agreed scope, following our approach and activities as outlined in the "Project overview" chapter between the 6th of July 2022 and the 09th of November 2022. Feedback from the Kolibri DAO development team was received and Inference performed a follow-up assessment.

Based on our scope and our performed activities, our security assessment revealed no security issues. However, different observations were made, which if resolved with appropriate actions, may improve the quality of the wrapper.

The follow-up assessment showed that all of the points raised, have been addressed accordingly.

In November 2022, the Kolibri DAO team reported a bug in the code and Inference reviewed the changes applied to the two smart contracts in scope to fix the bug. Based on our activities, we have not discovered any security issues or observations on the updated code. The reported bug was a functionality bug and no assets were in danger at any time.

Overview on issues and observations

Details for each reported issue or observation can be obtained from the "Security issues" and "Observations" sections.

Row header	Severity / Status
Issues	
There are no open, known security issues.	
Observations	
O-KDW-001: Coding	- / closed
O-KDW-002: Inefficiency	- / closed
O-KDW-003: No two-step procedure to replace admin address	- / closed
O-KDW-004: Documenting Compiler Version	- / closed
O-KDW-005: Sending tez to contracts	- / closed
O-KDW-006: Validity of information retrieved from other contracts	- / closed
O-KDW-007: Usage of latest token balance	- / closed

Project overview

Scope

The scope of the security assessment was the following set of smart contracts:

- Trustless-DEX-Market-Maker
  - quipuswap_maker_ceiling.py
    and the included files in order to compile the smart contract

  - quipuswap_maker_ceiling.tz
    which is the compiled Michelson code

- Quipuswap-Liquidity-Proxy
  - quipuswap_liquidity_proxy.py
    and the included files in order to compile the smart contract

  - quipuswap_liquidity_proxy.tz
    which is the compiled Michelson code

All files in scope were made available via a source code repos: https://github.com/chasdabigone/Trustless-DEX-Market-Maker and https://github.com/chasdabigone/Custody-Free-Quipuswap-Wrapper. Our initial security assessment considered commits "7470c7feea51a8c18c39e295f58c6af01a1a815a" and "4d55da468503a9ab35c71226e76ee85b07a199a4" respectively.

Later on, the repositories were merged for simplicity and consistency. The code is now stored in a new repository and commit hash "cabfa11cbad911f04b4c1a2c6414a30fbc4dd295" has been considered in our follow-up assessment: https://github.com/chasdabigone/Custody-Free-Quipuswap-Wrapper.

Additionally, we also had access to the following documentation in order to build up our understanding of the smart contracts:
https://github.com/chasdabigone/Custody-Free-Quipuswap-Wrapper/tree/main/docs

Our re-assessment for the updated version 1.1 of the two smart contracts in scope considered the changes to the code assessed in the follow-up assessment. Commit hash: "c0d62ed4fb9935c0e438ee57533ff89ff30b8664".

Scope limitations

Our security assessment is based on the following key assumptions and scope limitations:

- Any potential adversarial activities conducted by the administrator of the contract or operational errors by administrators were out of scope.
- Smart contracts for tokens not in scope were considered trusted.
- Key management of associated secret keys has not been assessed.
- Claims, implied by the use of the word DAO in the project's name, regarding the decentralised nature of the protocol/platform were out of scope.
- Analysis of impact of the economic situation including the appropriate settings of limits, trade amounts, etc. was out of scope.

Methodology

Inference’s methodology for smart contract security assessments on Tezos is a combination of a source code review of the smart contract source code in the high-level language (e.g. Ligo or SmartPy), and the resulting compiled code in Michelson. This approach provides additional assurance that the compiler producing the Michelson code is correct, and does not introduce any security issues. Furthermore, this approach fosters a better understanding for smart contract users of what has been reviewed and what has been effectively deployed on-chain.

In order to ensure a high quality in our security assessments, Inference is using subject matter experts having a high adversarial scenario mindset to spot potential issues in smart contracts under review. Additionally, we apply checklists derived from good practices and commonly known issues based on the Tezos smart contract assessment checklist to document our work and to ensure good issue coverage.

Furthermore, Inference maintains regular communications with the smart contract development team to ensure a correct understanding of the smart contract solution and environment, but also to make teams aware of any observations as soon as possible.

Inference’s internal quality assurance procedures ensure that results of security assessments are challenged for completeness and appropriateness by a second independent expert.

Objectives

The objectives are the identification of security issues with regards to the assessed smart contracts and their conceptual design and specification. The security assessment also focuses on adversarial scenarios on specific use cases which have been listed in appendix "Adversarial scenarios". These were identified together with the Kolibri DAO developers and checked during our security assessment.

Activities

Our security assessment activities for the defined scope were:

- Source code review of smart contract code written in SmartPy
- Source code review of the smart contract in Michelson
- Review of testing code

We applied the checklist for smart contract security assessments on Tezos, version 1.1 obtained from https://github.com/InferenceAG/TezosSecurityAssessmentChecklist. We applied the following security checklist tables:

- System / Platform
- Storage
- Gas issues and efficiency
- Code issues
- Transactions
- Entrypoint
- Admin / Operator functions
- Other topics & test cases

Our activities for the follow-up assessment were:

- Source code review of changes in the smart contract code in SmartPy
- Source code review of changes in the smart contracts in Michelson
- Reassessing security issues and observation from initial assessment in case they are claimed to be resolved

Our activities for the re-assessment of the version 1.1, which is fixing a functionality bug, were:

- Source code review of changes in the smart contract code in SmartPy
- Source code review of changes in the smart contracts in Michelson

Security issues

There are no open, known security issues.

Observations

O-KDW-001: Coding

The code is not easily readable, because in some circumstances function names and variable names are undescriptive. Key/Value assignments are not easily readable. Some error strings are unclear to what they refer to.

Both contracts in scope are affected.

Recommendation:
Good practice regarding code readability and simplicity should be followed. The naming and assignment convention of Python should be followed. The function “open_some” should always use error strings.

Comment from Kolibri DAO:
All of the points listed above will be addressed and fixed within the timeframe of the audit process.

Results from follow-up assessment:
The code has been updated and all points listed above have been addressed. Status changed from “open” to “closed”.

O-KDW-002: Inefficiency

The Michelson code repeats 1.) the loading of the same values from storage, 2.) calculating the same expressions.
Code for loading and calculating consumes blockchain storage, which has to be paid for during contract origination. In addition, increased gas costs result for each entrypoint call.

Values retrieved from views, maps, and generally new variables used multiple times, should be stored in a SmartPy local variable in case they are used multiple times. Michelson consumes the values the first time it is used and redoes the view call each time the value would be used again.

Both contracts in scope are affected.

Recommendation:
We recommend using the function “sp.local” (https://smartpy.io/docs/general/variables/) or “sp.compute” to define local variables, values retrieved from views and values used in multiple locations.

Comment from Kolibri DAO:
All of the points listed above will be addressed and fixed within the timeframe of the audit process.

Results from the follow-up assessment:
The code has been updated and points raised have been addressed accordingly. Status changed from “open” to “closed”

O-KDW-003: No two-step procedure to replace admin address

The entrypoints “setGovernorContract” and “setExecutorContract” allow setting a new address for the admin role on the smart contracts. After verification that the current admin initiated the call, the entrypoint directly updates the admin role with the newly provided address.

This poses a risk that the address for the admin role is set to a wrong or non-working address.

This observation affects all smart contracts in scope.

Recommendation:
We recommend implementing a two-step procedure to change any critical privileged addresses. In a first step the current privileged address proposes a new address, followed by a second step in which the newly proposed address accepts this proposal. The change of the critical privileged address is only done after the acceptance at the second step.

Comment from Kolibri DAO:
The Governor is intended to be the DAO contract, which can execute a lambda like a multisig, but it takes approximately 1 week to go through the (public) proposal to execution process. The Executor is intended to be a traditional multisig contract that will perform the time sensitive actions of adding liquidity and vetoing a baker.
The idea is that the Governor (DAO) controls everything that it is able to, but we need an Executor to add liquidity because the required amounts can change in the timelock. It is a similar situation with veto where the intended baker could be changed during the timelock.
In order to change the Executor, the DAO would go through the proposal and voting process to run a lambda that updates with the setExecutorContract function. The Governor would only be changed if the DAO migrated its contract, and it would use the same proposal+vote system.
I am not sure a contract based multi-step process makes sense in this case because the DAO proposal system allows time to independently test and verify the lambda, and we don't want to have to submit two proposals to change a setting

O-KDW-004: Documenting compiler version

Documentation should list the compiler and its version used. This helps identify and follow compiler issues in the future.

Recommendation:
The used compiler and its version should be listed in the project documentation.

Comment from Kolibri DAO:
The point above will be addressed and fixed within the timeframe of the audit process.

Results from follow-up assessment:
The documentation has been updated and made more clear. Status changed from “open” to “closed”.

O-KDW-005: Sending tez to contracts

Entrypoints which do not expect any tez, are not rejecting them. However, if tez is sent to the contract, the contract Governor is able to withdraw the sent tez by using the entrypoint “send” in the contract “quipuswap_liquidity_proxy.py”. Both contracts in scope are affected by this.

Recommendation:
We recommend failing an operation, in the case where the tez are not expected to be sent to an entrypoint. This should at least be done for any non-privileged entrypoint. This can easily be done in SmartPy using the entrypoint flag “check_no_incoming_transfer=True”.

Comment from Kolibri DAO:
The point above will be addressed and fixed within the timeframe of the audit process.

Results from the follow-up assessment:
The code has been updated and all appropriate entrypoints are now rejecting tez transfers. Status changed from “partly closed” to “closed”.

O-KDW-006: Validity of information retrieved from other contracts

The smart contract “maker_ceiling” retrieves price information from the Harbinger Oracle but does not check the age of the data.

Recommendation:
We recommend checking the validity of data with regards to stale or out of date parameters.

Comment from Kolibri DAO:
The point above will be addressed.

Results from the reassessment:
The code has been updated and the date of the retrieved data is being checked for staleness. Status changed from “open” to “closed”.

O-KDW-007: Usage of latest token balance

In the smart contract “maker_ceiling” the entrypoint “returnBalance” emits two operations, one to get the current balance from the token contract, and one to transfer tokens with the current balance stored within the local storage. The balance stored within the local storage is always updated after the transfer has been made and is thus not up to date.

Recommendation:
We recommend moving the transfer call within the entrypoint “redeemCallback”. The new flow would be: 1. Get the current token balance with the callback from the token contract, 2. Transfer the token according to the newly updated balance from the local storage.

Comment from Kolibri DAO:
The point above will be addressed.

Results from the reassessment:
The code has been updated. The flow has been changed according to our suggestion. Status changed from “open” to “closed”.

Disclaimer

This security assessment report ("Report") by Inference AG ("Inference") is solely intended for Tezos Foundation ("Client") with respect to the Report's purpose as agreed by the Client. The Report may not be relied upon by any other party than the Client and may only be distributed to a third party or published with the Client's consent. If the Report is published or distributed by the Client or Inference (with the Client's approval) then it is for information purposes only and Inference does not accept or assume any responsibility or liability for any other purpose or to any other party.

Security assessments of a software or technology cannot uncover all existing vulnerabilities. Even an assessment in which no weaknesses are found is not a guarantee of a secure system. Generally, code assessments enable the discovery of vulnerabilities that were overlooked during development and show areas where additional security measures are necessary. Within the Client's defined time frame and engagement, Inference has performed an assessment in order to discover as many vulnerabilities of the technology or software analysed as possible. The focus of the Report's security assessment was limited to the general items and code parts defined by the Client. The assessment shall reduce risks for the Client but in no way claims any guarantee of security or functionality of the technology or software that Inference agreed to assess. As a result, the Report does not provide any warranty or guarantee regarding the defect-free or vulnerability-free nature of the technology or software analysed.

In addition, the Report only addresses the issues of the system and software at the time the Report was produced. The Client should be aware that blockchain technology and cryptographic assets present a high level of ongoing risk. Given the fact that inherent limitations, errors or failures in any software development process and software product exist, it is possible that even major failures or malfunctions remain undetected by the Report. Inference did not assess the underlying third party infrastructure which adds further risks. Inference relied on the correct performance and execution of the included third party technology itself.

Appendix

Adversarial scenarios

The following adversarial scenarios have been identified and checked during our security assessment.

Scenario	Impact rating & descriptive	Assessment result
As a normal user, add myself as a governor/executor.	High	**Ok** Nothing identified that could circumvent the checks in the smart contract.
Change the contract's state from which it can not recover.	High - As the contract would become unusable.	**Ok** Nothing identified that could circumvent the checks in the smart contract. The contract only has one callback.

Risk rating definition for smart contracts

Severities are quantified with two dimensions, roughly defined as follows, whereas the examples have to be regarded as indication only:

Probability of occurrence / materialisation of an issue
(bullets for a category are linked with each other with “and/or” condition.)

- Low:
  - A trusted / privileged role is required.
  - Contract may end up in the issue if other conditions, which are also unlikely to happen, are required.
- Medium:
  - A specific role or contract state is required to trigger the issue.
  - Contract may end up in the issue if another condition is fulfilled as well.
- High:
  - Anybody can trigger the issue.
  - Contract’s state will over the short or long term end up in the issue.

Impact:
(bullets for a category are linked with each other with “and/or” condition.)

- Low:
  - Non-compliance with TZIP standards
  - Unclear error messages
  - Confusing structures
- Medium:
  - A minor amount of assets can be withdrawn or destroyed.
- High:
  - Not inline with the specification
  - A non-minor amount of assets can be withdrawn or destroyed.
  - Entire or part of the contract becomes unusable.

Severity:

	Low impact	Medium impact	High impact
High probability	**High**	**Critical**	**Critical**
Medium probability	**Medium**	**High**	**Critical**
Low probability	**Low**	**Medium**	**High**

Glossary

Term	Description
Ligo	High level smart contract language. Website: https://ligolang.org/
Origination	Deployment of a smart contract
SmartPy	High level smart contract language. Website: https://smartpy.io/
TZIP	Tezos Improvement Proposal