QuipuSwap v3 for Madfish

Tezos Foundation

Independent security assessment report

inference

Report version: 1.0 / date: 21.09.2023

Table of contents

Table of contents **2**
Summary **4**
Overview on issues and observations 5
Project overview **6**
Scope 6
Scope limitations 6
Methodology 7
Objectives 7
Activities 8
Security issues **9**
S-MQ3-001: gas exhaustion risk because of unbounded set 9
S-MQ3-002: Locked Pools when having huge prices 10
Observations **11**
O-MQ3-001: FA2 non-compliant balances 11
O-MQ3-002: use of abs() for integer conversion 12
O-MQ3-003: transfer from a valid operator is denied 13
O-MQ3-004: same-pair pool with different tick spacing cannot be deployed 14
O-MQ3-005: Factory does not reject unnecessary tez 15
O-MQ3-006: wrong computation for negative denominators 16
Disclaimer **17**
Appendix **18**
Adversarial scenarios 18
Risk rating definition for smart contracts 19
Glossary 20

Version / Date	Description
1.0 / 21.09.2023	Final version

Summary

Inference AG was engaged by the Tezos Foundation to perform an independent security assessment of Madfish's QuipuSwap v3.

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 10th of July 2023 and the 30th of August 2023. Feedback from Madfish was received and Inference performed a follow-up assessment.

Based on our scope and our performed activities, our security assessment revealed several low-risk security issues. Additionally, different observations were also made, which if resolved with appropriate actions, may improve the quality of QuipuSwap v3.

This report only shows remaining open or partly resolved issues and observations. Inference has received the following feedback from Madfish regarding all these reported matters.

Comment from Madfish:
Madfish acknowledges and agrees with all points raised. Inference clearly outlined all possible dangers and outcomes. Since the project is already live on mainnet and all issues are low priority Madfish decided not to proceed with fixes.

Overview on issues and observations

At Inference AG we separate the findings that we identify in our security assessments in two categories:

- Security issues represent risks to either users of the platform, owners of the contract, the environment of the blockchain, or one or more of these. For example, the possibility to steal funds from the contract, or to lock them in the contract, or to set the contract in a state that renders it unusable are all potential security issues;
- Observations represent opportunities to create a better performing contract, saving gas fees, integrating more efficiently into the existing environment, and creating a better user experience overall. For example, code optimizations that save execution time (and thus gas fees), better compliance to existing standards, and following secure coding best practices are all examples of observations.

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

	Severity / Status
Issues	
S-MQ3-001: gas exhaustion risk because of unbounded set	low / open
S-MQ3-002: Locked Pools when having huge prices	low / open
Observations	
O-MQ3-001: FA2 non-compliant balances	- / open
O-MQ3-002: use of abs() for integer conversion	- / open
O-MQ3-003: transfer from a valid operator is denied	- / open
O-MQ3-004: same-pair pool with different tick spacing cannot be deployed	- / open
O-MQ3-005: Factory does not reject unnecessary tez	- / open
O-MQ3-006: wrong computation for negative denominators	- / open

Project overview

Scope

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

- Factory
  - CameLIGO code: /contracts/main/factory.mligo, including all code & files to create the contract with entrypoint "main"
  - Michelson code: KT1JNNMMGyNNy36Zo6pcgRTMLUZyqRrttMZ4 on Tezos mainnet
- Pool:
  - CameLIGO code: /contracts/main/dex_core.mligo, including all code & files to create the contract with entrypoint "main"
  - Michelson code for the Pool TKEY/wTEZ (FA2/FA2): KT1BTdtJpJqRRTR3pKMvKU6QsmhwbKXf1rrM on Tezos mainnet

All files in scope were made available via a source code https://github.com/madfish-solutions/quipuswap-core-v3, and our security assessment considered commit “051e2a137800b1ca71ed9fd1712c0973b8bae2d1”.

Scope limitations

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

- Any potential adversarial activities conducted by the administrators/owners of the contract or operational errors by administrators/owners were out of scope.
- The key management of associated secret keys has not been assessed.
- Since anyone can deploy QuipuSwap v3 smart contracts and have the initial ability to configure any token pair, we did not assess the potential risks associated with users interacting with such deployments that could potentially involve invalid or malicious tokens.
- Analysis of impact of the economic situation including the appropriate settings of initial or default values such as limits, tick spacing, etc., but also approximations.

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 Madfish team 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 CameLIGO
- Source code review of the smart contract in Michelson

We applied the checklist for smart contract security assessments on Tezos, version 2.0 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
- On-chain views
- Admin / Operator functions
- Other topics & test cases

Security issues

S-MQ3-001: gas exhaustion risk because of unbounded set

The number of positions in the “positions_id” set for a user is not limited. This could lead to a gas exhaustion: if a user creates enough entries in the set, it is possible it becomes too expensive to deserialize when interacting with the smart contracts.

Probability - Low, since generating enough entries to cause a gas exhaustion issue requires a significant investment.
Impact - High.
Severity - Low, as the scenario is mathematically and economically unfeasible.

Recommendation:
Our recommendation is to limit the maximum size of entries a user can have in the set, to prevent a gas exhaustion issue.

S-MQ3-002: Locked Pools when having huge prices

The Pool smart contract calculates the square root price based on an index value, utilizing a helper lookup function called “ladder”. This ladder function facilitates the calculation of square root prices up to about 5.88e+22. However, if the square root price exceeds this limit, the lookup in the ladder function will fail, resulting in the Pool smart contract becoming locked.

Furthermore, the Pool smart contract does not take into account the decimals of a token in its calculation. Consequently, the price limit can potentially be reached if there is a significant disparity in the values of the handled tokens and/or if there is a significant difference in the decimals of tokens (25 digits or more).

Probability - Low, even in pools that have significant differences in decimal precision, this scenario requires extreme changes in prices.
Impact - High, the pool would be inaccessible and locked in its state once it reaches any of these extreme states.
Severity - Low, as the scenario is an extreme edge case.

Recommendation:
Our recommendation is to implement logic to assess the difference in precision for the tokens contained in the Pool smart contract.

Observations

O-MQ3-001: FA2 non-compliant balances

The FA2 standard (TZIP-012) defines that token balances should be stored in a standardized way to allow token balances to be indexed.

Recommendation:
We recommend modifying the indexing approach to comply with the specification.

O-MQ3-002: use of abs() for integer conversion

The use of the abs function to convert a value from an INT type to a NAT type is discouraged, as it can open up scenarios where negative values are incorrectly converted to NAT numbers.

The two following instances of the abs function were identified in the scope of the security assessment:

- Variable “time_passed” is converted using abs in “update_time_cumulatives”
- Value “10000 - fee_bps” is converted using abs in “one_minus_fee_bps”.

We have not identified any security issues in the current code, since the surrounding code where abs function is used ensures that parameters for the abs function are negative.

Recommendation:
We recommend revising the approach to avoid utilizing the abs function for converting INT types to NAT types. Instead, consider implementing a helper function that guarantees the absence of negative numbers as input. This approach safeguards against potential problems arising from future code modifications.

O-MQ3-003: transfer from a valid operator is denied

The Pool smart contract implements a check to prevent pointless transfers from happening. The check utilizes the following clause:

"tx.to_ = (Tezos.get_sender())"[1]

to check that the transfer is not an attempt from an user to transfer tokens from themselves to themselves.
However, if “sender” was an operator on behalf of “from”, this would represent a valid transfer that is incorrectly refused and reverted.

Recommendation:
We recommend modifying the evaluation to identify pointless transfers to take into account the possibility of transfers from operators, as described above.

1 https://github.com/madfish-solutions/quipuswap-core-v3/blob/051e2a137800b1ca71ed9fd1712c0973b8bae2d1/contracts/partial/token/fa2.mligo#L37

O-MQ3-004: same-pair pool with different tick spacing cannot be deployed

The Factory smart contract uses a key to identify the different pools deployed. This key does not keep track of "tick_spacing" as a primary identifier for a pool. This means that it is impossible to deploy a pool that uses the same token pair and the same "fee_bps", but a different tick spacing, as a previously deployed one, as there would be a primary key violation in the map that keeps track of existing pools.

Recommendation:
We recommend adding the tick spacing as a primary identifier for a pool, to allow more flexibility for users that want to create their own pools.

O-MQ3-005: Factory does not reject unnecessary tez

The Factory smart contract does not perform any check to verify that no tez has been sent along with the contract calls it receives. As the smart contract has no use for these tez, and it also does not implement any mechanism to return them to the original author of the contract call or to extract them from the contract, all tez sent to the Factory smart contract will be locked and lost.

Recommendation:
We recommend implementing a check for the whole Factory smart contract to refuse and revert contract calls that contain tez attached to them.

O-MQ3-006: wrong computation for negative denominators

The following function

let ceildiv_int (numerator : int) (denominator : int) : int = - ((- numerator) / denominator)

does not behave correctly in scenarios where *denominator* is a negative value, causing the overall computation to return inaccurate values.

We have not identified any security issues in the current code under review, since this ceildiv_int function is not feeded with negative value.

Recommendation:
Our recommendation is to implement an assert statement to ensure no wrong negative denominators are provided as parameters to the function.

Disclaimer

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 an admin.	High	**Ok** Nothing identified that could circumvent the checks in the smart contract.
Influence the price by swapping large amounts to gain an economical advantage.	High	**Ok** Nothing identified.
Deploy a pool with tokens that have a huge difference in precision, to draw users to it and cause it to reach an extreme price value to lock users' funds.	High	**Not ok** See our reported security issue **S-MQ3-002** While the attack vector is possible, the only gain would be to cause a financial loss to users of the pool. There is no economical gain for the attacker, making the attack scenario less feasible.
Add a huge amount of positions to the "positions_id" set to cause the contract to be too costly to deserialize, locking it and its funds forever.	High	**Not ok** See our reported security issue **S-MQ3-001** The impact is severe, but adding enough entries to make the attack plausible requires an economical cost that is unlikely to be sustained by any attacker, even more so taking into account that there is no economical gain to draw from this scenario.
Receiving more tokens in a token transfer than the correct calculated amount of tokens.	High	**Ok** Nothing identified.
Attaining a greater liquidity position than the accurately calculated amount based on the provided tokens.	High	**Ok** Nothing identified.
Acquiring more tokens than the liquidity position, including rewards, would allow.	High	**Ok** Nothing identified.

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