Ubinetic's improved flat curve for Youves

Tezos Foundation

Independent security assessment report

inference

Report version: 1.0 / date: 16.08.2023

inference

Table of contents

Table of contents **2**
Summary **4**
Overview on issues and observations 5
Project overview **6**
Scope 6
Scope limitations 7
Methodology 8
Objectives 9
Activities 9
Security issues **10**
There are no open known security issues. 10
Observations **11**
UIC-001: Influencing the liquidity token price 11
UIC-002: Lack of testing 12
UIC-003: Results of views potentially are not up to date 13
UIC-004: FA12 non-conformance 14
UIC-005: Race condition in “set_lqt_address” entrypoint 15
Disclaimer **16**
Appendix **17**
Adversarial scenarios 17
Risk rating definition for smart contracts 19
Glossary 20

Version / Date	Description
1.0 / 16.08.2023	Final version

Summary

Inference AG was engaged by the Tezos Foundation to perform an independent security assessment of Ubinetic's improved flat curve smart contracts.

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 6 of July 2023 and the 28 of July 2023. Feedback from the Youves team was received and Inference performed a reassessment.

Based on our scope and our performed activities, our security assessment revealed a few low to medium severity rated security issues. Additionally, different observations were made, which if resolved with appropriate actions, may improve the quality of Ubinetic's improved flat curve smart contracts.

All security issues have been resolved by the Ubinetic team. This report only shows remaining open or partly resolved observations.

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	
There are no open, known security issues.	
Observations	
UIC-001: Influencing the liquidity token price	- / open
UIC-002: Lack of testing	- / open
UIC-003: Results of views are not up to date	- / partially closed
UIC-004: FA12 non-conformance	- / open
UIC-005: Race condition in "set_lqt_address" entrypoint	- / open

Project overview

Scope

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

- tez cash
  - CameLIGO code: /contract/swap/tez_cash_flat_cfmm_with_rewards.mligo, including all code & files to create the contract with entrypoint "main"
  - Michelson code: __SNAPSHOTS__/compilation/swap/tez_cash_flat_cfmm_with_rewards_contract.tz[1]
- token cash
  - CameLIGO code: /contracts/swap/token_cash_flat_cfmm_with_rewards.mligo, including all code & files to create the contract with entrypoint "main"
  - Michelson code: __SNAPSHOTS__/compilation/swap/token_cash_flat_cfmm_with_rewards_contract.tz[2]
- liquidity pool
  - CameLIGO code: /contracts/swap/liquidity_pool.mligo
  - Michelson code: __SNAPSHOTS__/compilation/swap/liquidity_pool_contract.tz

All files in scope were made available via a source code repo: https://gitlab.papers.tech/papers/ubinetic/sap-smart-contract and our initial security assessment considered commit “feb675e4c8b0ba18298b24c6958251a309733cea”.

Our final reassessment considered commit: “15930ee3c45d020889587e65f71a3cad0”.

[1] Compiled by Ubinetic with set compiler switch “TOKEN_IS_FA2”
[2] Compiled by Ubinetic with set compiler switches “TOKEN_IS_FA2” and “CASH_IS_FA2”

Scope limitations

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

- Any potential adversarial activity 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.
- The economic consequences in the event of drastic economic situations (e.g. loss of peg) were not within the scope of our consideration.

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 Ubinetic team and checked during our security assessment.

Activities

Our activities for the initial assessment and for the defined scope were:

- Source code review of smart contract code written in CameLIGO

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

Our activities for the reassessment were:

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

Security issues

There are no open known security issues.

Observations

UIC-001: Influencing the liquidity token price

Due to the applied curve, any deviation from the point where the underlying token balances are in perfect equilibrium results in an increased liquidity token price.

The price, which can be determined using offered views or callback entrypoints, calculates the liquidity token price directly based on the current balances of the underlying tokens.

If another solution directly utilises this provided liquidity token price, the price could potentially be manipulated in favour of the user.

For instance, a user could temporarily manipulate the price by a flash attack[3]:

1. A user exchanges a remarkable amount of tokenA for tokenB in the improved flat curve smart contract. As a result, the generated imbalance causes an increase of the liquidity token price.
2. The same user could potentially exploit the higher liquidity token price offered by the improved flat curve smart contract in another contract that queries the liquidity token price from the improved flat curve smart contract.
3. Ultimately, the user exchanges tokenB back for tokenA

Recommendation:
We recommend either

- Calculating and providing the liquidity token price in a way which is not susceptible to such an attack. For instance, one approach is to offer a time-weighted price for the liquidity token.
- Clearly informing users about the liquidity token price, potential risks involved, and guidance on how to use it.

Comment from Ubinetic:
Whenever the LQT token will be used in other contracts, Youves will fetch the price via a time weighted price oracle contract.

[3] All operations are executed in one transaction. Thus, the operation emitter is not vulnerable to an arbitrage between its own set of emitted operations.

UIC-002: Lack of testing

The smart contracts in scope are lacking test cases.

A thorough testing process can greatly improve the confidence in the correct behaviour of the smart contracts.

Recommendation:
We recommend implementing tests for all newly developed functionalities.

Comment from Ubinetic:
We decided to only do integration testing for this. There are no unit tests for the contract.

UIC-003: Results of views potentially are not up to date

The smart contracts in scope implement views that might not always reflect the most recent data.

In a scenario where “update_pool” has not been executed for a while, the result of the views might return values that do not closely match the actual content of the pools in the smart contracts.

Recommendation:
We recommend making sure that users get up to date values, or at least that they are aware of the potential risk they take when relying on their return values.

Comment from Ubinetic:
We are aware that the views might not return the correct values because they might not have the latest balances.We are okay with it, and we named them lazy calculated prices.

Results from reassessment:
We updated the status from “open” to “partially closed”. We have not closed the issue to raise awareness of this situation among potential users of those views.

UIC-004: FA12 non-conformance

The liquidity pool smart contract is not compliant with the FA12 specification TZIP-007 as follows:

- The error message “NotEnoughBalance” is a value of type NAT instead of a construct of (nat :required, nat :present), where *required* is the requested amount of tokens, *present* is the available amount
- The error message “NotEnoughAllowance” is a value of type NAT instead of a construct of (nat :required, nat :present), where *required* is the requested amount of tokens, *present* is the current allowance
- The error message “UnsafeAllowanceChange” is a message of type STRING instead of the allowance value upon the contract call as of type NAT.

Recommendation:
We recommend complying with the specification, so that all components operating with these smart contracts can expect standardised behaviour from them.

Comment from Ubinetic:
We used the FA1.2 contract Ligo library provided by the Ligo community. As it is a matter of error type, we don't see this as a blocker.

UIC-005: Race condition in “set_lqt_address” entrypoint

The “set_lqt_address” can be called by everyone as long as the parameter “lqtAddress” has not been set yet. After setting the value, the entrypoint can no longer be used to set a new one.
Thus, depending on how smart contracts are originated and initialised, this may open a race condition where unauthorised third parties call the “set_lqt_address” entrypoint first.

Recommendation:
We recommend implementing a protection mechanism to prevent this race condition from happening, saving a potentially huge number of failed contracts to be deployed and wasted.

Comment from Ubinetic:
We are aware of such an issue, but the liquidity pool and the cfmm are interconnected and one must set the other, therefore we will need at least one operation after.
The cfmm is the admin of the liquidity pool contract and the cfmm must know what is the address of the liquidity pool.
In either case, there must be an operation after deploying the contract that would either:

- Set the address of the liquidity pool in the cfmm (currently happening).
- Set the cfmm as the admin of the liquidity pool

Disclaimer

This security assessment report (“Report”) by Inference AG (“Inference”) is solely intended for the 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	Assessment result
Front-running the initialization of the liquidity address "Set_lqt_address" has no access control in place. An attacker could always front-run the deployment and set the liquidity address to a wrong one, making it impossible to deploy a contract. It is always possible to deploy a new one, but as long as the attacker front-runs it, the scenario repeats.	**Not ok** Ubinetic replied that they are aware of this issue, and the approach taken is explained in UIC-005.
Integer division causes funds loss When adding liquidity, is there the possibility to deposit and not obtain anything in exchange? Due to the way integer division works, if a user deposits very little, maybe the return value is a fraction too small to return. The only protection in place is minLqtMinted, which does not have a strict range of admissible values, however.	**Ok** No scenario was identified, unless the pool is in a severely imbalanced state.
Low-precision tokens can lead to issues Due to the way division works, there is the possibility that two users depositing different amounts get the same return of deposited tokens. If precision is not large enough, depositing X or X+DELTA, with a small enough DELTA, makes no significant difference in the end result. This would mean that there is an optimal price to pay, and that any excess paid is held custody by the contract, benefiting the general liquidity without any immediate return for the user.	**Ok** Using high-precision tokens makes this scenario very unlikely.
Swap "cash token" to "token": Swapper is able to obtain more tokens for the provided cash tokens than these tokens are worth at the time of exchange.	**Ok** Nothing identified
Swap "cash token" to "token": Swapper is able to prevent paying (part of) the exchange fee.	**Ok** Nothing identified
Swap "token" to "cash token": Swapper is able to obtain more cash tokens for the provided tokens than these cash tokens are worth at the time of exchange.	**Ok** Nothing identified
Swap "token" to "cash token": Swapper is able to prevent paying (part of) the exchange fee.	**Ok** Nothing identified

Remove liquidity: LP is able to withdraw more “tokens” than available LP tokens.	**Ok** Nothing identified
Remove liquidity: LP is able to withdraw more “cash tokens” than available LP tokens.	**Ok** Nothing identified
Providing liquidity, if DEX is not in balance, brings any advantages due to an unfair “lqt” distribution preferring one currency.	**Ok** Nothing identified
Minting LQT tokens without providing any or enough liquidity.	**Ok** Nothing identified
Burning LQT tokens without having any or enough liquidity tokens.	**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
TZIP	Tezos Improvement Proposal