Madfish’s lending protocol Yupana

Tezos Foundation

Independent security assessment report

inference

Report version: 1.0 / date: 22.08.2022

Table of contents

Version / Date	Description
1.0 / 22.08.2022	Final version

Summary

Inference AG was engaged by the Tezos Foundation to perform an independent security assessment of Madfish's smart contracts for the lending protocol Yupana.

Inference AG performed the security assessment based on the agreed scope, following our approach and activities as outlined in the "Project overview" chapter between 20th February 2022 and 12th August 2022. Feedback from Madfish was received during the course of the assessment and Inference performed several follow-up assessments.

Based on our scope and our performed activities, our security assessment revealed several security issues rated from critical to low severity. Additionally, different observations were also made, which if resolved with appropriate actions, may improve the quality of the Yupana protocol.

This report only shows remaining open or partly resolved issues and observations.

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-MYU-001: Testing	- / partly resolved
O-MYU-002: Documentation	- / open
O-MYU-003: Sending tez to contracts	- / open
O-MYU-004: No two-step procedure to replace admin address	- / partly resolved

Project overview

Scope

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

- yupana-protocol-core
- yapfy
- wTEZ

Please see section Scope details in the appendix for a more precise scope description.

All files in scope were made available via several source code repos, whereas the following one is the one which references to the other used ones:
https://github.com/madfish-solutions/yupana-contracts

Our initial security assessment considered the commits:

- “d669986c4dc1126ff34b465a1e00a1d13029137d” for yupana-protocol-core
- “43a954bdcdbe2873e4e5dfa4442b373313f283e1” for yapfy
- “73f39f0e5964ead21f5dd1bb1a48073693922fb7” for wTEZ

Our final reassessment considered the commits:

- “07389aa7b350188e3c7a9e742a8c5276fc4a05b8” for yupana-protocol-core
- “2c713cb01404c69ee0433f012f60f616e12b5a1e” for yapfy
- “bb45d4aae710777cb96c87804579a67f95ecda31” for wTEZ

The Michelson code for the following Tezos mainnet contracts were also considered and assessed:

- KT1Rk86CX85DjBKmuyBhrCyNsHyudHVtASec for yToken
- KT1UpeXdK6AJbX58GJ92pLZVCucn2DR8Nu4b for wTEZ
- KT1EyM6GsgfPyGKh9Nec2PcDNNrEZVGXQp7w for price feed router
- KT1Dzy4pXKg55d1JiZuW37ZUPm4JSjRoAUoP for ctez parser
- KT1ESorZtvMK87j9zfRximZgSrjvs1a5kHLi for ubinetic old parser
- KT1BFNdvnrjdBnoxmjjvcMKEDYqrGmmb2kyg for harbinger parser
- KT1X9xm8DFz646HZpGLrtkcPqPxXDtGXGYXv for wTEZ parser
- KT1HCDoQWNzxQqgifW3B2DMBKZuhrDNqtsv4 for InterestRate uBTC
- KT1Tnz3RM7cbdauRh1kTvd4Nvbg95YzmFTwD for InterestRate tzBTC
- KT1VQvPTVVZDagCCBoQ2e68Zr89Xk6dwx46S for InterestRate uUSD

- KT1MEzk2RVSxKyokUMCPGmhnmnS5QQrcQKmd for InterestRate kUSD
- KT1NKusqCZ7HUeCCUtkg8gQgPDbVozgq2RVn for InterestRate cTez
- KT1Vtz3mawhqfReStHR6qSEv2k1sPLaDW9VC for InterestRate wTEZ

In addition to the readme in the different,, we also had access to the following documentation in order to build up our understanding of the Yupana lending protocol: https://yupana-finance.gitbook.io/yupana-document-portal/developer-space/.

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 were regarded as trusted, since we assumed that these smart contract addresses only get whitelisted by Madfish's team after passing their due diligence process.
- Economic model including the appropriate settings of thresholds, factors, etc. was out of scope. Also not in scope was assessing the impact of changing thresholds, factors, etc. when the protocol is live by using an admin function such as e.g. "setGlobalFactors", or "setTokenFactors".
- The oracles are deemed as trusted sources by Madfish. Thus, potential risks of a misbehaving price oracle have been 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 beens listed in appendix Adversarial scenarios. These were identified together with the Madfish developers and checked during our security assessment.

Activities

Our security assessment activities for the defined scope were:

- Source code review of smart contract code in Ligo
- Source code review of the deployed smart contract in Michelson
- Review of testing code

We applied the checklist for smart contract security assessments on Tezos, version 1.0 obtained from https://github.com/InferenceAG/TezosSecurityAssessmentChecklist. We applied the following security checklist tables:

- System / Platform
- Storage
- Gas issues and efficiency
- Code issues
- Transaction
- Entrypoint
- On-chain views
- 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 Ligo

- Source code review of changes in the deployed smart contracts 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

O-MYU-001: Testing

Various test cases are defined for the smart contracts in scope. However, we observed that some entrypoints are not covered by test cases at all and we also identified several test cases which should be added in order to fully cover the smart contract's function and its correctness.

This observation affects all smart contracts in scope.

For more details please see our reported results in our provided work documentation.

Recommendation:
We recommend covering all entrypoints and code with appropriate test cases.

Comment from Madfish:
Has been addressed.

Results from follow-up assessment:
Partly resolved. Missing important test cases have been added. However, there are still code parts which are not fully covered. Status changed from "open" to "partly resolved".

O-MYU-002: Documentation

Documentation is available, but quality and level of details varies for both the different smart contracts in scope and for entrypoints within the smart contracts.

For more details please see our reported results in the provided work documentation.

Recommendation:
We recommend improving the documentation. The documentation should be directed to smart contract users and should provide sufficient information about how the smart contracts work, on how to use them and also the risks associated with them, including suggestions of “Do’s and Don’ts” to mitigate those risks.

With regards to the entrypoints, we recommend that the documentation for every single entrypoint at a minimum includes:

- Description of what the entrypoint is for and what it should do,
- Meaning and types of input parameters / arguments,
- Pre-conditions (e.g.who can call the entrypoint and whether operator / approve permissions are required),
- Post conditions,
- Returned operations (transfers, contract origination, etc.).

Comment from Madfish:
We might add more documentation later, since it is not quintessential for the project to function properly.

O-MYU-003: Sending tez to contracts

Entrypoints which do not expect any tez to be sent do not reject any sent tez. Thus, if tez are mistakenly sent with a transaction to the smart contract then the sent tez are locked in the smart contract since there is no withdrawal function available.

This observation affects all smart contracts in scope.

Recommendation:
We recommend either rejecting any sent tez, in the case where the tez are not expected to be sent to an entrypoint, or consider implementing a privileged, access-restricted, entrypoint to withdraw any tez which have been accidentally sent.

Comment from Madfish:
Omitted.

O-MYU-004: No two-step procedure to replace admin address

The entrypoint “setAdmin” allows setting a new address for the admin role on the smart contracts. After verification that the current admin initiated “setAdmin”, the “setAdmin” entry point 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 the 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.

At the very least, we recommend analysing the situation, ensuring with appropriate documentation and procedures that admins dealing with these contracts are not mistakenly removing the last existing admin.

Comment from Madfish:
Fixed.

Results from follow-up assessment:
Partly resolved. Resolved for the yToken smart contract, but not for the priceFeedProxy and interestRate smart contracts.

Disclaimer

Appendix

Scope details

- yupana-protocol-core:
    - yToken
        - contracts/main/yToken.ligo and the included Ligo files in order to compile the smart contract
    - Interest rate
        - contracts/main/interestRate.ligo and the included Ligo files in order to compile the smart contract
- yapfy
    - Router
        - contracts/main/router.ligo and the included Ligo files in order to compile the smart contract
    - Parsers:
        - contracts/main/parser/ctez.ligo and the included Ligo files in order to compile the smart contract
        - contracts/main/parser/harbinger.ligo and the included Ligo files in order to compile the smart contract
        - contracts/main/parser/ubinetic.ligo and the included Ligo files in order to compile the smart contract
        - contracts/main/parser/ubinetic_old.ligo and the included Ligo files in order to compile the smart contract
        - contracts/main/parser/wtez.ligo and the included Ligo files in order to compile the smart contract
- wTEZ
    - contracts/main/fa2.ligo and the included Ligo files in order to compile the smart contract

Adversarial scenarios

The following adversarial scenarios have been identified together with Madfish developers and checked during our security assessment.

yToken:

Scenario	Impact rating & descriptive	Assessment result
Minter receives more share tokens than the correct amount.	Critical, due to • Emptying pool • Drainage	**Ok** Nothing identified.
Redeemer receives more tokens back than the correct amount.		**Ok** Nothing identified.
Borrower receives more borrowed tokens than the correct amount.		**Ok** Nothing identified.
Repayer has to pay back less than the correct amount.		**Ok** Nothing identified.
Liquidator can also liquidate accounts not open for liquidation.	High, due to • Loss of funds by borrower • Loss of reputation	**Ok** Nothing identified.
Liquidator can liquidate more than the defined limits.		**Ok** Nothing identified.
Liquidator can achieve a higher liquidate reward than the correct one.		**Ok** Nothing identified.
Minting/redeeming a specific token in the same block in order to fetch any rewards (new rewards or not yet distributed rewards) within the particular block. Note: Minting/redeeming in the same block does not occasion costs to the user (resp. only blockchain transaction costs).	Medium, since part of the rewards/profits are wrongly distributed.	**Ok** Nothing identified. *New rewards:* Rewards for Yupana users are collected and distributed if the “accrueInterest” EP is called, which augments the “totalBorrows” of the corresponding token with the accumulated interests, since the last call. Calling “accureInterest” multiple times in the same block for the same token does not have any impact, since the factor “blockDelta” is zero for each additional call. *Not distributed rewards:* Furthermore, calling mint EP requires that an “accrueInterest” has been done already within the same block. Thus, these rewards have already been distributed, before a user mints.

Borrowing/repaying a specific token in the same block in order to make any profit within the Yupana protocol. Note: Borrowing/repaying in the same block does not occasion costs to the user (resp. only blockchain transaction costs).	Medium, since part of the rewards/profits are wrongly distributed.	**Ok** Nothing identified. Generally, borrowing a token creates costs over time for the borrower. No costs are only the case if the repayment happens in the same block. Thus, a borrower could try to use the borrowed token to make any profit within the same block. However, we have not found any possibility to make these profits within the Yupana protocol itself.
Users are able to borrow any token without providing any collateral or only a fraction of the required collateral. (Note: free flash loans)	Critical, since the entire protocol gets imbalanced.	**Ok** Nothing identified to get a “free flash loan”. Borrowing a token requires corresponding collateral.

Interest rate:

Scenario	Impact rating & descriptive	Assessment result
Abusing the callback getter methods in order to provide wrong data to a target contract.	Depends on the use case. For the yToken: Feeding the yToken contract with wrong rates would be critical.	**Ok** This is generally possible, since data to calculate rates is provided by the calling contract and then delivered to the callback contract provided with the request. Thus, it is the responsibility of the contract, who needs data from the interestRate contract, that the contract ensures that the data has been requested and updated by himself. For the yToken contract: This is appropriately handled: - User has first to call “updateIntersest” EP. - updateInterest calls interestRate EP with its own parameters to calculate the “borrowRate”. - InterestRate contract calls back on “accrueInterest”. - accureInterst ensures the call comes from the interestRate contract and also updates locally a timestamp. - Furthermore and important the “updateInterest'' sets the lock “isInterestUpdating”, which then is removed by “accrueInterest”. This ensures the update has been requested by yToken. updateInterest fails if lock has not been set.

yapf:

Scenario	Impact rating & descriptive	Assessment result
Feed yToken priceCallback entrypoint with wrong price updates for a token.	High.	**Ok** Nothing identified. Price receiving entrypoints on yToken, router and parser contracts all implement checks whether update comes from authorized contract source.
Parser can be abused to send a price to the router for a token the parser is not responsible for.	High.	**Ok** Nothing identified. Each parser fails, if the provided tokenID in getPrice EP is not registered with the parser. cTez parser, which is the only parser having a receivePrice callback entrypoint can also not be abused to send a price update for another token upstream.
Router or parser responsible for multiple tokens commingles prices and tokenIds.	High.	**Ok** Nothing identified.

wTEZ:

Scenario	Impact rating & descriptive	Assessment result
Obtaining more wTEZ than the provided tez would give.	Critical, due to ● Emptying pool ● Drainage	**Ok** Nothing identified.
Obtaining more tez than the provided wTEZ would give.		**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