TezFin by StableTech

Tezos Foundation

Independent security assessment report

inference

Report version: 1.0 / date: 19.12.2023

inference

Table of contents

Version / Date	Description
1.0 / 19.12.2023	Final version.

Summary

Inference AG was engaged by the Tezos Foundation to perform an independent security assessment of TezFin by StableTech.

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 3rd of January 2023 and the 11th of December 2023. Feedback from StableTech was received and Inference performed a follow-up assessment.

Based on our scope and our performed activities, our initial security assessment revealed several security issues. Additionally, different observations were made, which if resolved with appropriate actions, may improve the quality of TezFin.

This report only shows remaining open or partly resolved issues and 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
Security issues	
There are no known open security issues.	
Observations	
O-TFI-001-01: FToken is TZIP-007 non-compliant	- / open
O-TFI-001-02: Approval not possible for new users	- / open
O-TFI-002: Gas exhaustion risks	- / open
O-TFI-003: Entrypoints and views access outdated data	- / open
O-TFI-004: Changing interest rate model can cause erroneous calculations	- / open
O-TFI-005: Failing on-chain views	- / open
O-TFI-006: Interest rates are based on block levels	- / open
O-TFI-007: Gas optimizations	- / open

Project overview

Scope

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

- Comptroller:
  - SmartPy code: contracts/Comptroller.py with compilation target “Comptroller”
  - Michelson code: compiled_contracts/Comptroller/step_000_cont_0_contract.tz and the lambdas stored in compiled_contracts/Comptroller/step_000_cont_storage.tz
- Interest Rate Model “FXTZ_IRM”:
  - SmartPy code: contracts/InterestRateModel.py with compilation target “CXTZ_IRM”
  - Michelson code: compiled_contracts/CXTZ_IRM/step_000_cont_0_contract.tz
- Oracle Proxy:
  - SmartPy code: contracts/TezFinOracle.py with compilation target “TezFinOracle”
  - Michelson code: compiled_contracts/TezFinOracle/step_000_cont_0_contract.tz
- Token contracts:
  - FUSDtz:
    i. SmartPy code: contracts/CFA12.py with compilation target “CUSDtz”
    ii. Michelson code: co mpiled_contracts/CUSDtz/step_000_cont_0_contract.tz and the lambdas stored in compiled_contracts/CUSDtz/step_000_cont_storage.tz
  - FBTCtz:
    i. SmartPy code: contracts/CFA2.py with compilation target “CBTCtz”
    ii. Michelson code: compiled_contracts/CBTCtz/step_000_cont_0_contract.tz and the lambdas stored in compiled_contracts/CBTCtz/step_000_cont_storage.tz
  - FXTZ:
    i. SmartPy code: contracts/CXTZ.py with compilation target “CXTZ”

ii. Michelson code: compiled_contracts/CXTZ/step_000_cont_0_contract.tz and the lambdas stored in compiled_contracts/CXTZ/step_000_cont_storage.tz

All files in scope were made available via a source code repo: "https://github.com/StableTechnologies/TezFin" and our initial security assessment considered commit "e1603bfc21617360b0915d9e364d7a8658f32775".

Our follow-up assessment considered commit: "76789d119d4930a34ef583086f3e831ecbdae51e".

Furthermore, we assessed the Michelson code for the following contracts on the Tezos mainnet:

Name	Contract address
Comptroller	KT1CF6EantmpPVfqdm9mDgsrMiFB2i81gcWn
FFA12_IRM	KT1TSYWvDnHjVj36GYdSrwCRJGEQpQoAbdMN
FXTZ_IRM	KT1Ax6eUUbPjvdfhroH5dcEtJx4eDt8hFHcq
FFA2_IRM	KT1CB8zyHFvsexa5cFExD7KXGCzRQpjaMXnX
TezFinOracle	KT1FU6obZqPu5S7c23R8iUzabDAuG47Jqsrm
FUSDt (FA2)	KT1GYKoownVC1ukP2TBDgKx7bSXRM5XkV1W6
FUSDtz (FA12)	KT1MX7D6ZJp2DDSSeDS96JPTFPXKkNiHFhwb
FXTZ	KT1W8P4ZxD8eREKjDnxMe5882NP3GnAgrv46

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.
- Key management of associated secret keys has not been assessed.
- Content of metadata and token metadata was out of scope.
- Data provider setup providing prices to the TezFin oracle was out of scope.
- The economic model of TezFin, including the appropriate settings for thresholds, factors, indexes, etc. was out of scope. Additionally, potential impacts of significant changes in the economic environment were not assessed.

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 StableTech 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

We applied the checklist for smart contract security assessments on Tezos, version 1.2 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

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
- Source code review of smart contracts in Michelson deployed on Tezos mainnet
- Reassessing security issues and observation from initial assessment in case they are claimed to be resolved

Security issues

There are no known open security issues.

Observations

O-TFI-001: Implementation of the TZIP-007 standard is faulty

There are two observations regarding the implementation of the TZIP-007 standard in the smart contracts under review:

O-TFI-001-01: FToken is TZIP-007 non-compliant

The error messages thrown by the FToken contract are not compatible with the FA1.2 (TZIP-007) standard.
Entrypoints affected by this issue include:

- “transfer”, with error codes “CT_TRANSFER_NOT_APPROVED” or “CT_INSUFFICIENT_BALANCE”
- “approve”, with error code “CT_UNSAFE_ALLOWANCE_CHANGE”

Moreover, ledger balances for FA1.2 tokens are not stored according to the TZIP-007 specification. In some cases, this could cause these balances to not be correctly indexed.

Recommendation:
We recommend following the specification for error messages of the FA1.2 (TZIP-007) standard.
We also recommend to follow the specification for the ledger storage design, to make sure that the tokens are appropriately indexed and shown in wallets.

Comment from StableTech:
We consider this a minor deviation at best and choose to keep the existing code. This will not affect compatibility with indexers, wallets and other contracts.

O-TFI-001-02: Approval not possible for new users

The implementation of the TZIP-007 token does not allow to “approve” users that don’t already have an existing balance in the “balance” big map.
The latest version of the FA1.2 SmartPy template accounts for this possibility with the following check:

spender_balance = self.data.balances.get(sp.sender, default=sp.record(balance=0, approvals={})

However, the correct behaviour to follow is not clearly specified in the standard.

Recommendation:
We recommend following the template implementation, to appoint operators by users with no existing balance.

Comment from StableTech:
The suggested behavior is superfluous to the needs of the protocol so we are skipping it.

O-TFI-002: Gas exhaustion risks

Several occurrences of gas exhaustion risks due to the deserialization of unbounded data structures or looping algorithms over data structures of unbounded size were identified:

- If a lender registers too many markets, the SET data structure used to track them can become too large to deserialize, locking the Comptroller in an inaccessible state. Also, the SET data structure used inside the big map to keep track of loans poses the same risk: if a user can have loans in too many markets, the data structure can grow too big to deserialize. However, the possibility to add markets is reserved to admins, so the risk for this specific instance is reduced;
- The loop in “calculateAccountLiquidityWithView” over collateral tokens and loans registered can run for an unpredictable number of iterations, potentially exceeding the gas limit;
- The loop in “updateAllAssetPrices” and in “accrueAllAssetInterests” over registered markets can run for an unpredictable number of iterations, potentially exceeding the gas limit.

However, since all above outlined points depend on the number of registered “markets”, which only can be added / changed by admins, we rate this point as an observation and not as a security issue.

Recommendation:
We recommend implementing a limit mechanism for all the scenarios above, to limit the size of data structure and make sure that loops do not have to process an unfeasible number of items.

Comment from StableTech:
The number of markets won’t foreseeably exceed a few dozen and the administrators can ensure that proper bounds are maintained so that there is never any gas exhaustion risk.

O-TFI-003: Entrypoints and views access outdated data

The following entrypoints in the FToken smart contract do not ensure that the values they require are up to date:

- getBalanceOfUnderlying
- borrowBalanceStored
- borrowBalanceStoredView
- exchangeRateStored
- getCash

Moreover, callers of these entrypoints cannot predict, based on the received callback, whether the received data was computed using up-to-date information.

Additionally, the following views do not ensure that the returned data is up-to-date:

- liquidateBorrowAllowed
- liquidateCalculateSeizeTokens

Recommendation:
We recommend returning an additional parameter, to indicate that the returned value may not be up-to-date.

Comment from StableTech:
This is by design and we will ensure that all public documents ensure that an interest accrual must be done as part of any TezFin action. The protocol incentives ensure that all parties have reason to comply and trigger an interest accrual before performing any action.

O-TFI-004: Changing interest rate model can cause erroneous calculations

The “setInterestRateModelInternal” functionality allows to change the interest rate model of the smart contracts. However, this functionality does not calculate all the previous interest rates before changing the model.
This means that, if values are not forcefully calculated before making the change, all old interests that have not been computed yet will be calculated with the new interest rate model, which represents unknown behaviour to potential users of the platform and a source of uncertainty for them.

However, changing the interest rate model can only be done by admins, we rate this as an observation and not as a security issue.

Recommendation:
When the decision to change the interest rate model at a theoretical block X has been made, we recommend to first calculate all interests up to block X and then initiate the interest model change.
In order to do so, admins could ensure the contract is in the correct state before calling “setInterestRateModel”.

Comment from StableTech:
The protocol administrators would handle accruals correctly as part of any prudent rate model change.

O-TFI-005: Failing on-chain views

Several on-chain views fail instead of returning a flag to signal their failure. This will cause all contracts relying on them to fail as well, without giving them the opportunity to handle the situation themselves.

The following failing views were identified:

- “seizeAllowed” in Comptroller
- “getAccountSnapshotView” in FToken

Recommendation:
We recommend to either return a flag value (0, None...) to signal that the view has not functioned correctly, or to thoroughly document this failing behaviour implemented.

Comment from StableTech:
After considering the tradeoffs, we decided to maintain the existing behavior.

O-TFI-006: Interest rates are based on block levels

The interest rate uses block levels to measure the passing of time and compute the growth of interests accrued. This approach does not take into account the possibility that a new protocol, with a different block time, is implemented, or a possible downtime period of the blockchain that could alter the expected block time.

Recommendation:
We recommend considering the scenarios of a new chain protocol being implemented and the potential effects of a different block time on the interest computations performed by the smart contracts.

Comment from StableTech:
In a decentralized and distributed system, there is no assured way to keep timestamps synchronized. If Tezos ever decides to change the way time is handled, we will raise this point as part of the conversation.

O-TFI-007: Gas optimizations

The Michelson review of the compiled smart contracts highlighted some areas of improvement that could save gas costs during execution, as shown in the following summary.

TezFinOracle

- Function “isAdmin” could be a lambda

FXTZ_IRM

- Function “utilization_rate” could be a lambda
- Function “calculateBorrowRate” could be a lambda

FXTZ, FUSDtz, and FBTCtz

- Function “getBorrowBalance” could be a lambda
- Function “exchangeRateStoredImpl” could be a lambda
- Variable “repayAmount” could be a sp.local

Comptroller

- Variable “pricePair” could be a sp.local
- Variable “params” in “calculateAccountAssetLiquidityView” could be a sp.local
- Function “updateAllAssetPrices” could be a lambda

Recommendation:
We recommend modifying the smart contracts accordingly, to improve the performance and the efficiency of the code and save on gas costs.

Comment from StableTech:
After considering the tradeoffs, we decided to maintain the existing behavior.

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
A user is able to add himself to the admin list.	High	**Ok** Nothing identified that could circumvent the checks in the smart contract.
A user is able to transfer tez out of the contract.	High	**Ok** Nothing identified that could circumvent the checks in the smart contract.
Locking the contract in an inaccessible state.	High	**Ok** Although possible, the investment required to lead the smart contracts to gas exhaustion is a deterrent big enough to make this attack unfeasible, for the current state of the chain and its environment.
Influencing the governance process to alter the interest rate model to the attacker's favour.	High	**Ok** Nothing identified that could circumvent the checks in the smart contract.
Minting more NFT than allowed in the market.	High	**Ok** Nothing identified that could circumvent the checks in the smart contract.
Spending the tokens of other users without prior approval.	High	**Ok** Nothing identified that could circumvent the checks in the smart contract.
Liquidating a borrow that is not available for liquidation.	High	**Ok** Nothing identified that could circumvent the checks in the smart contract.
Altering or influencing the results from the oracles to obtain more favourable rates.	High	**Ok** Nothing identified that could circumvent the checks in the smart contract.

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