P2P Lending and Borrowing from OpusDei

Tezos Foundation

Independent security assessment report

inference

Report version: 1.0 / date: 07.09.2022

inference

Table of contents

Version / Date	Description
1.0 / 07.09.2022	Final version

Summary

Inference AG was engaged by the Tezos Foundation to perform an independent security assessment of OpusDei's P2P Lending and Borrowing.

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 9th of June 2022 and the 16th of August 2022. Feedback from OpusDei was received and Inference performed a follow-up assessment.

Based on our scope and our performed activities, our security assessment revealed one security issue of critical severity and one of medium severity. Additionally, different observations were also made, which if resolved with appropriate actions, may improve the quality of OpusDei's P2P Lending and Borrowing.

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	
S-OLB-001: Gas exhaustion for loans, deals, and allowed tokens	critical / closed
S-OLB-002: Failing of 0 tez transaction	low / closed
Observations	
O-OLB-001: Documentation	-/ closed
O-OLB-002: Testing	- / partly resolved
O-OLB-003: Coding	-/ closed
O-OLB-004: Inefficiency	- / closed
O-OLB-005: Sending tez to contracts	- / closed
O-OLB-006: Compiler version	- / closed

Project overview

Scope

The scope of the security assessment was the following set of smart contract(s):

- P2P Lending and Borrowing
  - contract written in SmartPy: opd/blob/main/contract.py
  - compiled contract in Michelson: opd/blob/main/contract.tz

All files in scope were made available via a source code repo: https://github.com/opdev3/opd and our initial security assessment considered commit "22bc26a808bf1fd0690a14e86b287dec463f4e25".

No additional documentation was provided at that time.

Our reassessment considered commit: "afb212a3068c56b4879869b096e5630a737645c4".

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 out of scope.
- 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, etc. when the protocol is live by using an admin function such as e.g. "set_min_deposit".

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 OpusDei 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 1.0 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
- 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 contract code in Michelson
- Reassessing security issues and observation from initial assessment in case they are claimed to be resolved

Security issues

S-OLB-001: Gas exhaustion for loans, deals, and allowed tokens

Information about loans, deals, and allowed tokens are stored in a “map” data type structure in storage.

This poses a risk, should these 2maps grow too large, that the storage can no longer be successfully loaded when an entrypoint is executed, since loading of a large storage may consume all available gas.

Probability - High. Depending on the popularity of the platform, users may create a lot of deals and loans, which may stay for a long time in the contract's storage.
Impact - High: The smart contract would become no longer callable and thus unusable. Tez provided as collateral would be locked in the smart contract.
Severity - Critical

Recommendation:
We recommend using "big_maps" instead of "maps" for "loans", "deals", and "tokens".

Comment from OpusDei:
Fixed.

Results from follow-up assessment:
Storage types have been changed to “big_map”. Status changed from “open” to “closed”.

S-OLB-002: Failing of 0 tez transaction

The entrypoint “setMinDeposit” allows setting “self.data.mindeposit” to zero and thus allows creating loans with 0 tez as collateral. Since sending out transactions with 0 tez results in a failing operation, this leads to:

- loans, which can not be cancelled using “cancelLoan” or
- deals, which can not be closed using “closeDeal”.

Probability - Low, since deals/loans with 0 tez do not really make sense.
Impact - Low: Neither collateral (0tez) nor borrowed tokens are locked in the contract. Borrowers can not use the “closeDeal” entrypoint in order to return the borrowed token, but could return the borrowed tokens to the owner by doing a direct token transfer to the owner.
Severity - Low

Recommendation:
We recommend not emitting any transaction, if the amount to be transferred is 0 tez.

Comment from OpusDei:
We did not assume collateral of 0 at the contract administration level. Now it's reworked, and the issue is fixed.

Results from follow-up assessment:
The code in the corresponding entrypoints have been rewritten. Status changed from “open” to “closed”.

Observations

O-OLB-001: Documentation

Documentation is missing. The contract's entry points and their parameters are described insufficiently. Code comments are also completely absent.

Recommendation:
We recommend adding documentation/specification which at least provides information about:

- Description what an entrypoint is for and what it should do
- Meaning and types of input parameters / arguments
- Pre conditions (e.g.who can call the entrypoint)
- Post conditions
- Returned operations (transfers, contract origination, etc.)
- Use case and workflow for users to use the smart contract even if the front-end website is not available.

Comment from OpusDei:
End user documentation is added as a Readme file on the project's GitHub. The file includes the main definitions, the glossary, a description of Peer-to-Peer services and their benefits, step by step user guide and other general information.

Results from follow-up assessment:
Documentation in the form of code comments, Use-cases, and workflow have been added. Status changed from "open" to "closed".

O-OLB-002: Testing

With regards to testing we observed the following points:

A. SmartPy testing (off-chain) only tests a very few failing conditions. Very often missing are failing tests e.g. in order to check whether verify checks work appropriately.
B. No on-chain testing in order to e.g. detect any compiler failures.

Recommendation:
We recommend defining and executing test cases in SmartPy appropriately covering all entrypoints. In addition to the testing in SmartPy, we also advise to define appropriate on-chain testing.

Comment from OpusDei:
We've done tests for all errors caught in the code. Done edge or corner cases in the input data checks and tests for all code branches inside checked functions.
The SmartPy off-chain testing was repeated on-chain, and the results were what the team expected. No protocols were written.

Results from follow-up assessment:
Off-chain test cases have been added. Status changed from “open” to “partly resolved”, since on-chain testing has not been added to the source code repository under review.

O-OLB-003: Coding

Our observations regarding coding style, good practice, and language usage:

- error codes are sparse and undescriptive, good practice regarding type checking are not being followed and
- the syntax of the Python naming convention is not being followed.

Recommendation:
We recommend considering:

A. Use more error codes. One of the three you use, is used everywhere and non descriptive.
B. Consider always using sp.set_type for entrypoints.
C. Follow the Python naming convention.

Comment from OpusDei:
Understood.

Results from follow-up assessment:
Recommendations have been implemented. Status changed from “open” to “closed”.

O-OLB-004: Inefficiency

The Michelson code repeats the loading of the same values from storage and calculating the same values. For instance, the variable “loan” in the entrypoint “cancelLoan” is used multiple times and not only once. Every time associate values are loaded again from the storage.

Repeating code to load variables from storage and to calculate values does consume a lot of blockchain storage, which has to be paid for during contract origination. In addition, increased gas costs result for each entry point call due to larger storage consumption than necessary and due to excessive calculations.

Recommendation:
We recommend using the function “sp.local” (https://smartpy.io/docs/general/variables/) or “sp.compute” to define local variables in SmartPy.

Comment from OpusDei:
Noted. Using sp.local everywhere it is possible.

Results from follow-up assessment:
The code has been updated to use local variables as much as needed. Status changed from “open” to “closed”.

O-OLB-005: Sending tez to contracts

Entrypoints which do not expect any tez to be sent do not reject any sent tez. However, if tez is sent to the contract, contract admins are able to withdraw the sent tez by using the entrypoint “withdraw”.

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.

Comment from OpusDei:
Noted and done.

Results from follow-up assessment:
Transactions with unexpected tez are now failing. Status changed from “open” to “closed”.

O-OLB-006: Compiler version

The documentation does not contain information about the SmartPy compiler version used to compile the Michelson code in the source code repo.

Recommendation:
We recommend adding information which SmartPy compiler version has been used to compile the Michelson code (contract.tz).

Comment from OpusDei:
Fixed.

Results from follow-up assessment:
Documentation has been added. 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 together with the OpusDei developers and checked during our security assessment.

Scenario	Impact rating	Assessment result
As a normal user, add myself as an admin.	High	**Ok** Nothing identified that could circumvent the checks in the smart contract.
Add a loan with ambiguous values.	High	**Ok** Nothing identified. All values are checked.
Withdraw all available tez, while loans are open to be accepted.	High	**Ok** Withdrawing respects the values of open loan requests.
Borrower withdraws collateral while not returning borrowed tokens or not all of them.	High	**Ok** Nothing identified.
Lender collects collateral or part of it before the granted loan expires.	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