Tezos Domains' governance, token, and vesting smart contracts

Tezos Foundation

Independent security assessment report

inference

Report version: 1.0 / date: 22.06.2023

inference

Table of contents

inference

Version / Date	Description
1.0 / 22.06.2023	Final version

Summary

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

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 12th of April 2023 and the 13th of June 2023. Feedback from the development team was received and Inference performed a reassessment.

Based on our scope and our performed activities, our security assessment highlighted some observations, which if resolved with appropriate actions, may improve the quality of the project.

This report only shows remaining open or partly resolved observations.

Overview on issues and 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 open, known security issues.	
Observations	
O-TZD-001: Lacking documentation	- / partially closed
O-TZD-002: Unused function	- / partially closed
O-TZD-003: Single step change procedure for critical address	- / partially closed
O-TZD-004: Potential deadlock situation	- / partially closed

Project overview

Scope

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

- **token-contracts:**
  - /single_asset/ligo/src/fa2_single_asset_1step_admin.mligo including all files in order to build the smart contract with entrypoint “main” and its Michelson code /artifacts/fa2_single_asset_1step_admin.tz
  - /single_asset/ligo/src/fa2_single_asset_delegated.mligo including all files in order to build the smart contract with entrypoint “main” and its Michelson code /artifacts/fa2_single_asset_delegated.tz

  in https://gitlab.com/tezos-domains/token-contract. The original commit reviewed was “81ff5f14ae4241c336f27a3bf673ca6bbd034c00”, the final commit reviewed was “fc6beee3609da6f519f3954365299a5a0fc7a5e1”
- **vesting-contracts:** /contracts/Vesting.mligo including all files in order to build the smart contract with entrypoint “main” and its Michelson code /artifacts/Vesting.tz in https://gitlab.com/tezos-domains/vesting-contract. The original commit reviewed was “c1605dfc543f3271d3a00ddc8317243ce9d9fbfe”, the final commit reviewed was “2af3b2bc257fbe7f2724d7b8f062c22dfc54f8a5”
- **governance-pool-contracts:** /contracts/GovernancePool.mligo including all files in order to build the smart contract with entrypoint “main” and its Michelson code /artifacts/GovernancePool.tz in https://gitlab.com/tezos-domains/governance-pool-contract. Our reviewed commit was “8dc348682fbea62a038379ff7c3600f4fb5bd6d8”.

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.

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 development 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

Our activities for the reassessment were:

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

Security issues

There are no open known security issues.

Observations

O-TZD-001: Lacking documentation

The documentation is not always thorough. While most of the functionalities are straightforward to understand, it would be advisable to improve the overall quality of the documentation for less experienced users. It would be a major improvement to include more detailed descriptions for:

- The general use case of each contract, including potential scenarios
- The description of each entrypoint and how it is supposed to behave
- The description of all parameters, admissible value ranges, example scenarios...

This applies to all contracts in scope.

Comment from the development team:
Improving the documentation will be addressed after the launch.

O-TZD-002: Unused function

The functions “mint_tokens” and “burn_tokens” (from token-contract) both use a function called “nop_operator_validator_function”. However, this function does not perform any action. Removing it would reduce the gas consumption of the contract.

Is it possible to remove it or does it have a purpose? Is it to comply with a specific function signature already present in another project?

Comment from the development team:
Yes it seems that the original code from https://github.com/oxheadalpha/smart-contracts did this to allow reusing the transfer function for mint and burn. We did not rewrite the original code for optimal gas cost because of the risk of introducing new bugs or vulnerabilities. Extending an existing codebase as opposed to rewriting it or writing completely from scratch was a deliberate choice.

Results from reassessment:
We agreed with the development team to add this observation to the report as a note to potential users of the platform.

O-TZD-003: Single step change procedure for critical address

While the general concept of the token-contract is to have this design in place, it is our duty as a security assessment company to note that this approach is considered insecure: a mistake during this procedure leads to severe risks, like having an unknown address as the new administrator of the contract (incorrectly typing the address during the change procedure), or changing administrator to an address whose private key has been lost.

Recommendation:
It would be more appropriate to implement a two-step change procedure for critical addresses. In this specific scenario, first suggest a new administrator's address, and then the new address should accept the role, proving that its keys are not lost and it can perform its new role as administrator.

Comment from the development team:
We understand the consequence of not having a confirmation mechanism, however we think it is necessary to be able to "throw away the keys", so to speak.

Results from reassessment:
We agreed with the development team to add this issue to the report as a note to potential users of the platform, to make them aware of the possible risk.

O-TZD-004: Potential deadlock situation

Our security assessment identified a potential deadlock situation that would render the governance-pool contract unusable. The following paragraph details how this deadlock can happen.

If a pool starts empty, as it supposedly should, with no votes and size equal to zero:

1. If a user deposits at the start of a pool's existence, there are no accrued rewards, so the pool size is zero. This means that attempting to deposit would cause a division by zero, that reverts the contract call
2. If a user deposits at a later point than the start, the pool has now a size bigger than zero. However, the number of minted votes is still zero. When the user attempts to deposit, the variable that keeps track of the minted votes will remain stuck at zero. It will also attempt to perform a 0-token transfer from the contract to the sender

Recommendation:
The code should take into account this possibility and prevent the contract from running into the described scenario.

Comment from the development team:
We initially considered handling the special case of 0 in both `pool_size` and `votes_minted` but eventually decided to simply initialize the pool with a non-zero amount (a virtual amount which will never be withdrawable, because nobody holds the vote tokens). This simplifies the contract logic.

There is an adjacent issue of what the initial value should be. If the value is very low (i.e. 1/1 for `pool_size` and `votes_minted`), once rewards are enabled without any deposits in the pool, the exchange rate becomes extremely large and natural number arithmetics we use will cause unacceptable rounding errors (consider an exchange rate of 1e9:1 after one day of rewards in an extreme example).

We initially explored three options:

1. Making the initial pool configuration something larger, like 1e9/1e9). The downside is that there are going to be rewards lost to this "virtual deposit".
2. Making the initial configuration 1/1 but depositing some initial tokens ourselves (before the rewards are enabled) that we don't withdraw. Basically a softer version of #1.
3. Adding specific logic that disables rewards altogether if the pool size is too low (for example the pool size is lower than the daily reward rate). This could be added to the accrual logic.

We eventually opted for number 2 for it's simplicity. It adds no complexity to the contracts or to the off-chain code that needs to interpret reward calculations.

Results from reassessment:
We agreed with the development team to add this issue to the report as a note to potential users of the platform, to make them aware of the possible risk and let them know what solution has been taken in order to prevent it.

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	Assessment result
Interact with the contracts to leave it in an unusable state.	**Ok** See O-TZD-004: Potential deadlock situation. An appropriate choice of the starting values of a pool lcan prevent this scenario from happening.
governance-pool-contract: More votes can be minted than the deposit would grant.	**Ok** Nothing identified.
governance-pool-contract: More tokens can be withdrawn than the assigned ones.	**Ok** Nothing identified.
vesting-contract: More tokens can be claimed than the assigned ones.	**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