Ubinetic's oracle smart contracts

Tezos Foundation

Independent security assessment report

inference

Report version: 1.0 / date: 12.08.2022

inference

Table of contents

Version / Date	Description
1.0 / 12.08.2022	Final version

inference

Summary

Inference AG was engaged by the Tezos Foundation to perform an independent security assessment of Ubinetic's oracle 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 10th July 2022 and 12th August 2022. Feedback from Ubinetic was received and Inference performed a follow-up assessment.

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 Ubinetic's oracle smart contracts.

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-UOR-001: Testing	- / open

Project overview

Scope

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

- Job scheduler:
  - contracts/oracle/job_scheduler.py and the included SmartPy files in order to compile the smart contract with the “JobScheduler” as a compilation target
  - compiled contract in Michelson: __SNAPSHOTS__/compilation/oracle/all/JobScheduler/step_000_cont_0_contract.tz
- Generic oracle v3
  - contracts/oracle/generic_oracle_v3.py and the included SmartPy files in order to compile the smart contract with the “PriceOracle” as a compilation target. Including the aggregation and validation lambda code which are in generic_oracle_v3.py as comments.
  - compiled contract in Michelson: __SNAPSHOTS__/compilation/oracle/all/GenericOracleV3/step_000_cont_0_contract.tz

All files in scope were made available via the source code repo https://gitlab.papers.tech/papers/ubinetic/sap-smart-contract and our initial security assessment considered commit “a705acd045b9cbed5e61cc8da0010326cf05eb1f”.

The following files from the source code repo were also part of the assessment:

- Testing files under folder tests/oracle/ and its included files in order to perform the defined tests.

Additionally, we also had access to the following documentation in order to build up our understanding of the Ubinentic oracle: https://ubinetic.com/oracles/

Our reassessment considered commit: a2f7b4771cbd77e39cf629af50402f5ed35bc660.

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.
- Data transmitters are trusted. Thus, data transmitters send data to the job_scheduler correctly and timely.

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 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 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 1.1 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
- 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-UOR-001: Testing

The smart contracts in scope have several test cases defined in SmartPy. However, we noted:

- The entrypoints “propose_admin” an “set_admin” in the job scheduler smart contract are not covered,
- The oracle generic v3 smart contract is not covered with test cases at all.

We also noted that, for both smart contracts in scope, there are no on-chain test cases available.

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 Ubinetic:
We are in the process with ECAD Labs to automate deployments and testing using Taqueria.

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 Ubinetic developers and checked during our security assessment.

Scenario	Impact rating & descriptive	Assessment result
Non-authorized data transmitters can submit prices.	High: Wrong prices provided by oracle	**Ok** The smart contract is checking whether "SOURCE" is a data transmitter. Thus, the source has to be an authorized data transmitter. However, since the data transmitter sends their fulfil requests to a "jobscheduler", there may be an attack scenario where it is possible that data transmitters submit their updates to a rogue "jobscheduler" acting as a Man-in-the-Middle. However, this depends on the setup of the data transmitters which is out of the scope of this assessment.
New oracle price is set even though not enough different data transmitters submitted their prices.	High: Wrong prices provided by oracle	**Ok** Not possible. The smart contract correctly checks whether the "threshold" of data transmitters has been reached. See also the next scenario's result with regards to the used "list".

Data transmitter subsequently submits its prices for the same epoch in order to get through its own prices.	High: Wrong prices provided by oracle	**Ok** Not possible. A data transmitter can only submit one price (set) per epoch. This is ensured in the smart contract, since a list can not contain multiple entries of the same data transmitter address. From the Michelson documentation for update of sets with b = True: *#If b is True then the value x is added to the set s. If x was already present in s, then the original set is returned.* Thus, a single data transmitter can not reach the defined threshold.
A single data transmitter can influence the oracle price “massively” (massively = deviation to the real actual price is more than 1/(2^10)).	High: Wrong prices provided by oracle	**Ok** Not possible. The first transmitter for an epoch sets the price that the other data transmitters are compared to. Thus, if a data transmitter wants to push a lower/higher price than the real actual price, the data transmitter has to be the first transmitter in an epoch and, furthermore, has to submit a price which is only lower/higher by ~1/(2^10) than the real actual price. The other data transmitters will then submit their prices which are close to the real actual price.
A number of data transmitters, not reaching the threshold, can influence the oracle price “massively” (massively = deviation to the real actual price is more than 1/(2^10)).	High: Wrong prices provided by oracle	**Ok** Not possible. As soon as an insufficient number of data transmitters are submitting a massively deviated price there is no quorum and thus no new price is set.

A single data transmitter or a number of data transmitters, not reaching the threshold, can block data price updates.	Medium: No price updates, which would only be a serious problem in case of volatile markets.	**Note** This is possible. If the first submitted price in an epoch from a data transmitter deviates massively from the real actual price, the smart contract will not get a quorum and thus no new price will be set. Thus, a rogue data transmitter may try to be the first data transmitter in each epoch in order to prevent oracle price updates. This issue is mitigated by the requirement that all data transmitters be white-listed a priori which reduces the attack surface to a malicious agent taking over a white-listed data transmitter.
The price can change more than (0.0625 * price) in one single epoch.	High: Wrong prices provided by oracle	**Ok** Not possible.
A data transmitter can lock the oracle for a longer time period.	Medium: No price updates, which would only be a serious problem in case of volatile markets.	**Note** This is possible if the data transmitter submits a future timestamp.

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