Token-gated video stream smart contract by Joko

Tezos Foundation

Independent security assessment report

inference

Report version: 1.0 / date: 22.12.2022

inference

Table of contents

Version / Date	Description
1.0 / 22.12.2022	Final version

Summary

Inference AG was engaged by the Tezos Foundation to perform an independent security assessment of the Token-gated video stream smart contract by Joko.

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 29th of September 2022 and the 07th of December 2022. Feedback from the Joko team was received and Inference performed a follow-up assessment.

Based on our scope and our performed activities, our security assessment revealed several security issues. Additionally, several observations were also made which, if resolved appropriately, may improve the quality of the assessed smart contracts.

Based on our activities in the follow-up assessment in this report we only list security issues and observations which have not yet been resolved by the Joko team.

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
Security issues	
There are no open, known security issues.	
Observations	
O-JPJ-001: Inefficient code	partially closed
O-JPJ-002: Usage of inappropriate types	open
O-JPJ-003: Testing	partially closed

Project overview

Scope

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

- Joko: contracts/joko.py with compilation target "Joko"
- FA2: contracts/joko.py with compilation target "FA2_comp"

All files in scope were made available via a source code repo:
"https://gitlab.com/MihirK018/joko-streaming-platform" and our initial security assessment considered commit "1830eaaf044db850293f7d261d8efb7433c11d3b".

Additionally, we also had access to the following documentation in order to build up our understanding of the smart contracts in scope:

- contracts/docs/Joko user work flow.pdf
- contracts/docs/joko_smart_contract_audit_documentation.pdf
- contracts/docs/Smart Contract Documentation.pdf

Our reassessment considered commit:
"6a32740713cb8b11d34e7754129f1e1e0fe38ce3"

In our reassessment, we also reviewed the Michelson code for the two smart contracts in scope:

- Joko: compilation/Joko/step_000_cont_0_contract.tz
- FA2: compilation/FA2_comp/step_000_cont_0_contract.tz

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. For instance: any off-chain views have not been 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 Joko 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
- 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 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-JPJ-001: Inefficient code

Parts of the SmartPy code for the Joko smart contract are not written efficiently leading to redundant Michelson code.

Redundant Michelson code consumes blockchain storage to store the code which has to be paid for during contract origination. Furthermore, increased gas costs are incurred for each entrypoint call due to larger than necessary storage consumption.

Recommendation:
We recommend the following:

1) using the function “sp.local” or “sp.compute” to define local variables which are used multiple times in the code.
2) restructuring the code by putting code into functions and considering using lambda functions where code elements are large or used very often.

Comment from Joko:
We are aware of some variables used multiple times that should be used by "sp.local" or "sp.compute".

Result from follow-up assessment:
sp.local has been used in several places. There are however remaining occurrences of the issue. Therefore, the Status is updated from “open” to “partially closed”.

O-JPJ-002: Usage of inappropriate types

The Joko smart contract is using the type “INT” instead the type “NAT” for variables which can not be a negative number.

Using inappropriate types for variables may lead to security issues where some characteristics of the type used are not correctly handled in the code. For instance, using an “INT” type instead of a “NAT” type could cause issues where defined boundaries / limits are bypassed by submitting negative numbers.

We raise this point as an observation and not as a security issue, since we have not identified any problems with the code in scope of this assessment.

Recommendation:
We recommend using appropriate types for any parameters. In particular, we recommend using “NAT” for the following variables:

- “max_mint_tier2” and “max_mint_tier3” in entrypoint “add_artist”
- “amount_tokens” in entrypoint “mint_JOKO_tier1”
- “amount_tokens” in entrypoint “mint_JOKO_tier2”
- “amount_tokens” in entrypoint “mint_JOKO_tier3”
- “tier2_max_mint” and “tier3_max_mint” in entrypoint “upate_max_mint”
- “max_mint_tier2” and “max_mint_tier3” in big_map “artist_map”
- big_map value in big_map “max_per_address”

Comment from Joko:
Keep type of some variable as sp.TInt as we already ensure these variables value greater than 0 so the value can not be 0 or be negative.

O-JPJ-003: Testing

The smart contracts in scope have several test cases defined in SmartPy however, the test cases do not cover every single aspect of the smart contract (limits, if statements, etc.).

Furthermore, we observed that there are no tests defined and executed to check the correctness of the smart contracts on-chain. Since SmartPy uses its own, different, Michelson interpreter than the real one implemented on-chain, there is a risk that contracts behave differently on-chain than contracts which have been tested with the SmartPy testing framework.

This observation affects all smart contracts in scope. However, since we reviewed the Michelson code for the two smart contracts in scope we raise this as an observation only.

Recommendation:
We recommend defining and executing appropriate test cases in SmartPy covering every single aspect of the smart contract. These tests should also appropriately verify whether the contract's state has been correctly changed and the contract is behaving as expected.

Furthermore, we highly recommend defining and implementing on-chain testing for smart contracts.

Comment from Joko:
Beside the tests made in the smartpy code, we do on-chain testing using better-call-dev based on black-box method. On-chain testing covers all the aspects that could possibly happen when interacting with Joko smart contract.

Result from follow-up assessment:
Several test cases have been added. However, not yet every single aspect is covered and on-chain testing is missing. Status updated from "open" to "partially closed".

Disclaimer

Appendix

Adversarial scenarios

The following adversarial scenarios have been identified and checked during our security assessment.

Scenario	Impact rating & descriptive	Assessment result
Users can bypass the implemented blacklist.	Medium - No direct financial impact.	**Ok** Nothing identified.
Users can put themselves into the whitelist without paying the required fee or not the full amount thereof.	High - Payment circumvention	**Ok** Nothing identified.
Users can obtain more tokens than the defined maximum.	Medium - No direct financial impact as long as they have to pay for it.	**Ok** Nothing identified.
Users can obtain tokens without paying the required fee nor the full amount thereof.	High - Payment circumvention	**Ok** Nothing identified.
As a normal user, I try to add myself as an admin.	High - Validity of the smart contract would not be a given anymore and the contract would have to be redeployed.	**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