AngryTeenagers by EcoMint

Tezos Foundation

Independent security assessment report

inference

Report version: 1.0 / date: 06.12.2022

inference

Table of contents

Version / Date	Description
1.0 / 06.12.2022	Final version

inference

Summary

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

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 October 2022 and the 06th of December 2022. Feedback from EcoMint 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 AngryTeenagers.

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

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-EAT-001: allow_list and pre_allowlist	- / open
O-EAT-002: Fund redirects	- / open

Project overview

Scope

The scope of the security assessment was the following set of smart contracts and the included SmartPy files in order to build the smart contract defined in the corresponding compilation target:

- NFT: main/nft_main.py
- sale: main/sale_main.py
- DAO: main/dao_main.py
- opt_out_voting: main/opt_out_voting_main.py
- majority_voting: main/majority_voting_main.py

All files in scope were made available via a source code repo: "https://github.com/eco-mint/angry-teenagers-contracts" and our initial security assessment considered commit "3387081697272d876f4ca945b91095e5b5cb42ca".

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

- doc/Angry_Teenagers_DAO_contracts.pdf
- doc/Angry_Teenagers_NFT_and_sale_contracts.pdf

Our reassessment considered commit: "896801a9b077f07871a77891832de6b370d46536" and the following deployed smart contracts on Tezos Ghostnet:

- KT1Sf89iosS6S91xoAfES54hxCtQw9GShkoY for NFT
- KT1Bn27CBu6F8FDH7HJHZ5rdqcLR7XR6LpR4 for sale
- KT1XZMJ27ZsCZiBDwrBkpAJi8z4vmBykxuKb for DAO
- KT1WHRvBmyvdXctxXfpmuJ8eQqf9zizR4kZs for opt_out_voting_main
- KT1DTCrDNZeAVQcHGSx21sgN3nEVwvxvLHjk for majority_voting_main

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.
- Claims, implied by the use of the word DAO in the project's name, regarding the decentralised nature of the protocol/platform 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 EcoMint 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-EAT-001: allow_list and pre_allowlist

The sale smart contract allows storing a list of addresses allowed to buy NFTs at certain stages in the sales process. These lists, “allowlist” and “pre_allowlist”, are stored in the smart contract storage as a set (Michelson type SET[1]).

This poses a risk: should this set grow too large the storage can no longer be successfully loaded when any entrypoint is executed since the loading of a large storage may consume all available gas.

However, since adding addresses to these two lists can only be done by the defined admins, we rate this point as an observation and not as a security issue.

Comment from EcoMint:
Allow list and pre-allowlist size are fully controlled by the EcoMint admin. They will be kept in a reasonable size.

[1] https://tezos.gitlab.io/michelson-reference/#type-set

O-EAT-002: Fund redirects

The sale smart contracts redirects received funds in the entrypoints “mint_pre_sale”, “mint_public_sale”, “pay_to_enter_allowlist_priv”, and “pay_to_enter_allowlist_pub” to the address defined in the variable “multisig_fund_address”.

These entrypoints fail if the address in “multisig_fund_address”is a smart contract where the default entrypoint fails if any tez are sent to it.

The fund redirects do not fail in the case where the destination is a smart contract accepting tez in its default entrypoint.

Based on information received from EcoMint we understand that the destination will be a smart contract with a default entrypoint accepting tez. Thus, we rate this point as an observation and not as a security issue.

Comment from EcoMint:
In our ecosystem, the variable multisig_fund_address always points to a multisig smart contract (the generic “Tezos multisig”). This contract defines the default entrypoint which accepts Tez to be transferred when called. No issue.

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 mint NFTs without being on the admin/sale admin list.	High: Voting without paying.	**Ok** Nothing identified that could circumvent the admin checks in the smart contract.
A user is able to add himself to the admin list.	High: Arbitrary write privileges to storage	**Ok** Nothing identified that could circumvent the checks in the smart contract.
A user is able to transfer tez out of the contract.	High: Admin fees and sales would be collectable.	**Ok** Nothing identified that could circumvent the checks in the smart contract.
A user is able to get more voting power without buying NFTs.	High: Voting without paying.	**Ok** Nothing identified that could circumvent the checks in the smart contract. Admins can “mint” tokens / voting power without paying for it, if an event is not open.
A user is able to end a voting period before the deadline.	High: Vote manipulation.	**Ok** Nothing identified that could circumvent the checks in the smart contract. The initial entrypoint is callable by anyone, but both voting strategies are comparing the current block height to the defined end block.
A user is able to buy NFTs without being on the allowlist.	High: Vote manipulation.	**Ok** Nothing identified that could circumvent the checks in the smart contract.

A user is able to enter the allowlist without paying for it.	Medium: Admins lose out on fees	**Ok** Nothing identified that could circumvent the checks in the smart contract.
A user on the allowlist is able to buy before the sale starts.	High: Vote manipulation.	**Ok** Nothing identified that could circumvent the checks in the smart contract.
A user is able to buy more NFTs/ voting power than allowed per user.	High: Vote manipulation.	**Ok** Nothing identified that could circumvent the check, if a new user is minting max per user or a recurring user is trying to mint more than max per user.
A user is able to add new tokens.	High: Vote manipulation.	**Ok** Nothing identified that could circumvent the admin 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