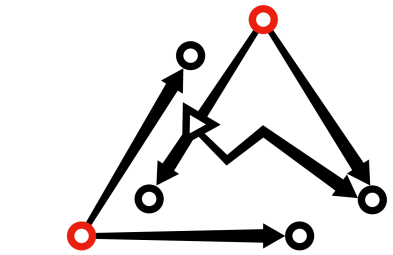

Least Authority

PRIVACY MATTERS

Kickflow Smart Contracts

Security Audit Report

Tezos Foundation

Final Audit Report: 8 October 2021

Table of Contents

Overview

Background

Tezos Foundation has requested that Least Authority perform a security audit of the Kickflow Smart Contracts. Kickflow is an open-source, decentralized crowdfunding and grants platform on the Tezos blockchain.

Project Dates

- **September 13 - September 24**: Code review *(Completed)*
- **September 29**: Delivery of Initial Audit Report *(Completed)*
- **October 7**: Verification Review *(Completed)*
- **October 8**: Final Audit Report delivered *(Completed)*

Review Team

- Phoebe Jenkins, Security Researcher and Engineer
- ElHassan Wanas, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Kickflow Smart Contracts followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

The following code repositories are considered in-scope for the review:

- Kickflow Funding Contracts: https://github.com/kickflowio/funding-contracts
- Kickflow FlowDAO: https://github.com/kickflowio/flow-dao

Specifically, we examined the Git revisions for our initial review:

Kickflow Funding Contracts: `05bd7f6e4cc400bf49f5f68b7699ed53d505998c`

Kickflow FlowDAO: `601b9acbe2d363db6adafe65b64417b330d13322`

For the verification, we examined the Git revision:

Kickflow Funding Contracts: `7fd8ea59ad48c4018a76de513868ef3cba4364c3`

Kickflow FlowDAO: `661b9e78d94b63b64c9cdfacdf9d8bdbc95a814d`

For the review, these repositories were cloned for use during the audit and for reference in this report:

https://github.com/LeastAuthority/funding-contracts

https://github.com/LeastAuthority/flow-dao

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third party code, unless specifically mentioned as in-scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- Kickflow Documentation: https://docs.kickflow.io/
- Medium Article, "Kickflow — Liberal Radicalism comes to Tezos": https://kickflowio.medium.com/kickflow-liberal-radicalism-comes-to-tezos-90508e8b5bbe
- V. Buterin, Z. Hitzig, E.G. Weyl, 2018, "Liberal Radicalism: A Flexible Design For Philanthropic Matching Funds." https://ssrn.com/abstract=3243656

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Adversarial actions and other attacks on the smart contract;
- Attacks that impacts funds, such as the draining or the manipulation of funds;
- Mismanagement of funds via transactions;
- Denial of Service attacks and security exploits that would impact or disrupt the execution of the smart contract;
- Vulnerabilities in the smart contract code;
- Protection against malicious attacks and other ways to exploit the smart contract;
- Inappropriate permissions and excess authority; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

The Kickflow platform is built on the principles of Quadratic Funding and Capital-constrained Liberal Radicalism (CLR) matching. It conducts periodic match funding rounds, whereby at the beginning of each round, entrants seeking funding submit entries for each matching round and receive an optimal share of the sponsor-funded pool. The amount they receive is based on the amount of funds the entry raised and the number of donations received.

The Kickflow smart contracts consist of the FlowDAO and Kickflow Funding Round smart contract implementations. The FlowDAO smart contracts are a DAO built on Tezos and utilize an FA1.2 governance token, by which token holders use the token to create or vote on proposals to set the parameters for and initiate a funding round. The Kickflow Funding Round smart contracts are deployed upon a FlowDAO approval of a new funding matching round proposal, which enables the funding round to be performed on-chain.

We found that security has been strongly considered in the overall Kickflow smart contracts system design and implementation. This is demonstrated by sound design decisions and trade-offs, in addition to the adherence to development best practices, as detailed below. This is particularly demonstrated by clean code, thorough testing, and extensive documentation, which provides clear examples of smart contract usage and increases confidence in the codebase.

System Design and Implementation

Our team performed a broad and comprehensive review of the FlowDAO and Kickflow Funding Round smart contracts and checked each set of smart contracts for system design flaws and implementation errors.

FlowDAO Smart Contracts

The FlowDAO FA1.2 governance token closely adheres to the FA1.2 token specification and we did not identify any implementation errors. While we found there is a complexity implemented in the FA1.2 governance token (i.e. a modification from the FA1.2 specification) by which the balance history of participants is stored, we acknowledge that this design decision is necessary to help in the prevention of flash loan attacks.

The FlowDAO smart contracts implement verification logic that checks that the user balance is the balance used rather than the immediate balance prior to execution of a vote or proposal, effectively preventing flash loan attacks. This verification requires storage of historical balances, which increases storage and gas costs, but hardens the security of the system. We found this to be a worthwhile design trade-off, as the protection against flash loan attacks helps to prevent attackers from being able to force arbitrary code through the FlowDAO smart contracts.

However, the FlowDAO smart contract proposals do allow the submission of arbitrary code via lambdas. These lambdas are relatively locked-down, by being of type `Unit`, and are intended to be reviewed by governance participants in the voting process. As a result, we did not identify any vulnerabilities in this design and implementation.

Kickflow Funding Round Smart Contracts

The Kickflow Funding Round smart contracts control the funding process upon approval from the FlowDAO smart contracts for a new funding round. The `donationhandler.py` smart contract confirms that donations received in funding rounds only come from whitelisted addresses. While the management of this whitelist could potentially benefit from a more decentralized implementation as a FlowDAO smart contract proposal to modify the whitelist, the current implementation follows a hierarchical structure for management where administrators can transfer ownership or add governors, and governors have control over the whitelist. The administrators are added through the FlowDAO smart contract, and transfers of administrator status within the smart contracts follow a recommended two-step confirmation process.

Our team closely reviewed user interactions with the funding pool and found that the Kickflow Funding smart contracts implement measures such that only sponsors are able to deposit funds before a funding round starts. Entrants can perform withdrawals of a non-zero CLR match or deposit only after the challenge period has ended. This eliminates possible scenarios where an incorrect control flow would lead to an early withdrawal by a malicious entry.

Finally, we found that the Kickflow Funding smart contracts do not perform any subtle calculations that would necessitate additional floating point math. Usage of floating point arithmetic is a common source of errors for pools. The Kickflow team has avoided the use by simply using the `Nat` type to represent amounts, equivalent to `mutez` or standard amounts of FA1.2 tokens. We commend the Kickflow team for taking this approach in an effort to mitigate against potential errors of this nature.

Code Quality

Our team conducted a manual review and analysis of the Kickflow smart contracts, including the FlowDAO and Kickflow Funding Round smart contract codebases. The code is commendable in that it is well-written and organized, adhering to development best practices. The division of behavior between smart contracts is well thought out and articulated, and components are logically organized such that each smart contract is logical on its own with little repetition. In some instances, we found some longer functions, however, they are accurately and thoroughly explained in the code comments facilitating a clear understanding of their purpose. This aided our ability to successfully reason about the security of the implementation.

However, we identified some instances that would benefit from error reporting. As a result, we recommend that error messages be provided for certain calls in order to help users identify and remediate errors (Suggestion 1).

Tests

We found that sufficient test suites have been implemented, providing adequate test coverage for the smart contracts. In particular, each smart contract contains extensive SmartPy tests, which help identify implementation errors and check that each component functions as intended. In addition, the tests are well-organized into smaller functions that are able to run independently of each other.

Documentation

The Kickflow team provided high-level overview documentation detailing the system architecture. In addition, both the FlowDAO and Kickflow Funding Round smart contract repositories include a documentation folder, which details technical information on the functionality and on interacting with the smart contracts. We found the documentation available for this review to be accurate, comprehensive, and helpful in supplementing our understanding of the system design and implementation.

Code Comments

The documentation contained within the code is sufficient and explains the intended functionality of each of the components. Larger functions contain in-line code comments at each logical block, which facilitates understanding each step of a function's execution.

Scope

Our team found that the scope of this security audit was sufficient in that it covered all security critical components of the FlowDAO and Kickflow Funding Round smart contracts.

Dependencies

As previously noted in SmartPy implementation security audits conducted by our team, the SmartPy compiler has not been audited by an independent security auditing team and that the development and maintenance of SmartPy is not conducted according to accepted security practices. As a result we suggest that the SmartPy compiler and CLI be reviewed by an independent security auditing team. We acknowledge that concerns regarding SmartPy are out of scope for the review and out of the control of the Kickflow team. However, SmartPy's security is critical for the security of any system utilizing SmartPy in its implementation. As such, we recommend the Kickflow team continue to monitor SmartPy developments and research. It is worth noting that SmartPy is effectively a deployment dependency, so potential changes will only be relevant to future deployments of components of the Kickflow smart contracts.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Suggestion 1: Provide Additional Error Messages for Certain Calls	Resolved

Suggestions

Suggestion 1: Provide Additional Error Messages for Certain Calls

Location

Examples (non-exhaustive):

kickflow-funding-contracts/donation_handler.py:46

kickflow-funding-contracts/donation_handler.py:109

Synopsis

Calls to the `open_some()` method of SmartPy `TOption` values have the potential to fail if the value is None. There is no error message implemented to make the cause for failure more obvious (such as in the examples, not having a stored address for a whitelist contract).

Mitigation

We recommend that the Kickflow team provide error messages to `open_some()` calls that may fail. Since some conditions have already been verified in certain functions, such as the presence of the address for the matching round contract, these can safely be left alone as they will not fail at that point in the function.

Status

The Kickflow team has added error messages that now clearly indicate the cause for failure.

Verification

Resolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. Although we are a small team, we believe that we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present, creating Issue entries, and for each we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Responsible Disclosure

Before our report or any details about our findings and suggested solutions are made public, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for resolution that balances the impact on the users and the needs of your project team. We take this agreed timeline into account before publishing any reports to avoid the necessity for full disclosure.