Tetragaze

Smart contract audit

Tetragaze

July 19, 2021 | v 1.0

Pass

Tetragaze Security Team has concluded that there are no issues that can have an impact on contract security. The contract is well written and is production-ready.

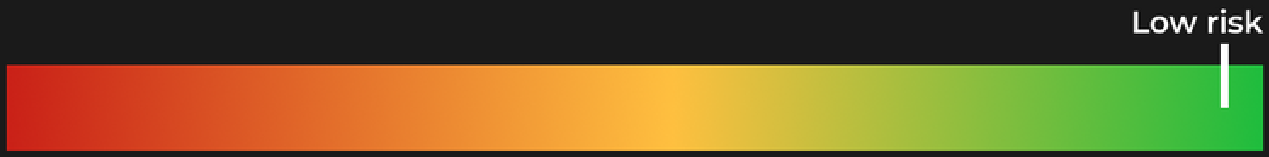
Score

90

Technical summary

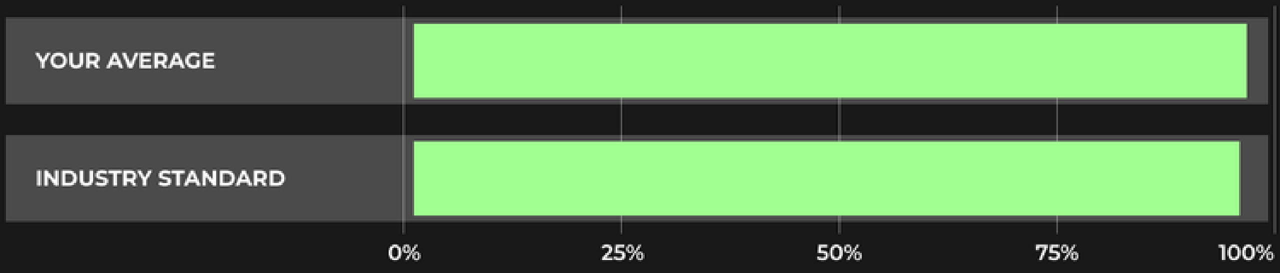
This document outlines the overall security of the Kucoin Launchpad smart contracts, evaluated by Tetragaze Blockchain Security team.
The scope of this audit was to analyze and document the Kucoin Launchpad smart contract codebase for quality, security, and correctness

Contract Status



There were no critical issues found during the audit.

Testable Code



Testable code is 90%, which is close to the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Tetragaze recommend that the Kucoin Launchpad team put in place a bug bounty program to encourage further and active analysis of the smart contract.

Table of content

Auditing Strategy and Techniques Applied

Requirements: FIP8 spec

During the review process, we made sure to carefully check that the token contract:

- Adheres to established Token standards and ensures effective implementation;
- Ensures that documentation and code comments accurately reflect the logic and behavior of the code;
- Ensures that token distribution is in line with calculations;
- Utilizes best practices to efficiently use gas, avoiding unnecessary waste;
- Employs methods that are safe from reentrance attacks and immune to the latest vulnerabilities;
- Follows best practices for code readability

Tetragaze team of expert pentesters and smart contract developers carefully evaluated the repository for security issues, code quality, and compliance with specifications and best practices. They conducted a line-by-line review, documenting any issues that were identified during the process. Special care was taken to ensure that the review was thorough and comprehensive.

1	Due diligence in assessing the overall code quality of the codebase.	2	Cross-comparison with other, similar smart contracts by industry leaders.
3	Testing contract logic against common and uncommon attack vectors.	4	Thorough, manual review of the codebase, line-by-line.

Summary

The security of the Kucoin Launchpad platform was evaluated in this report through the use of an audit procedure. During the audit, various issues of different levels of severity were identified. These issues were subsequently addressed and resolved by the auditee.

Structure and Organization of Document

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged "Resolved" or "Unresolved" depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

● Critical

The issue affects the ability of the contract to compile or operate in a significant way.

● High

The issue affects the ability of the contract to compile or operate in a significant way.

● Medium

The issue affects the ability of the contract to operate in a way that doesn't significantly hinder its behavior.

● Low

The issue has minimal impact on the contract's ability to operate.

● Informational

The issue has no impact on the contract's ability to operate.

Complete Analysis

Loops in the Contract are extremely costly

Medium

The release() function contains a for loop that iterates over the length of a non-memory array using a state variable as the loop condition. This causes the state variable to consume a significant amount of extra gas for each iteration of the for loop.

Recomendation:

It is advisable to use a local variable within a loop instead of accessing the length property of a state variable.

Missing error message for require functions.

Low

The require function on line 61 is missing an error message. It checks if the sender has sufficient balance to transfer the specified amount, and if that amount is greater than zero. It also checks if adding the amount to the recipient's balance will not exceed their balance.

```
require (balances[_from]>=_amount && allowed[_from]
[msg.sender]>=_amount && _amount>0 &&
balances[_to]+_amount>balances[_to]);
```

Recomendation:

It is suggested that an error message be included with each of the require functions mentioned above.

Complete Analysis

msg.sender is not checked

Low

Before continuing with further processing, it is necessary to validate that the msg.sender exists within the lockedAccount.

Recomendation:

We suggest using the require function to verify if the sender of the message (msg.sender) has been added to the lockedAccount list. This will prevent potential errors or unexpected behavior, as well as save gas, by ensuring that only valid values are processed.

Missing Check for Reentrancy Attack

Low

The following functions do not perform zero address validation:

- transfer()
- transferFrom()
- approve()
- timelock()
- mint()
- burn()

Recomendation:

To prevent unexpected behaviors and wasted gas, it is recommended to include require statements to validate all user-controlled input, including in the constructor, by implementing appropriate require statements. This will ensure that erroneous values do not affect the program.

Complete Analysis

State Variable Default Visibility

Low

The above-mentioned variable has no defined visibility. Clearly stating the visibility helps to prevent incorrect assumptions about who can access the variable.

It is assumed that state variables are internal by default, but it is important to explicitly specify this.

```
mapping (address=>uint256) balances;
mapping (address=>mapping (address=>uint256)) allowed;
mapping(address => LockedAccounts) lock;
```

Recomendation:

It is suggested to specify the visibility for the following variables as public, internal, or private. It is important to clearly define the visibility for all state variables.

Public function that could be declared external

Low

The functions timelock(), release(), and lockedAccountDetails() are never called by the contract and should be declared as external in order to save gas.

Recomendation:

For functions that are not called from within the contract, the external attribute should be used.

Complete Analysis

Not using delete to zero values

Low

The release() function includes the step of setting the following variables to zero:

- loc.amount = 0;
- loc.time = 0;
- loc.lockedAt = 0;

Recomendation:

Consider using delete to simplify the code and clearly convey the intended purpose.

Missing docstrings

Low

Due to the lack of documentation, it is extremely challenging to find contracts or functions. This lack of documentation hinders reviewers' understanding of the code's purpose, which is crucial for accurately assessing both security and correctness.

In addition, functions are more readable and easier to maintain when they are accompanied by docstrings. It is essential that functions are documented so that users can understand their purpose, potential failure points, who is permitted to call them, the values they return, and any events they emit.

Recomendation:

It is important to document all functions that are part of the public API of the contracts, including the parameters of each function. Additionally, even if they are not public, functions that perform sensitive tasks should also be thoroughly documented. For writing clear and concise docstrings, it is recommended to follow the Ethereum Natural Specification Format (NatSpec).

Complete Analysis

Use double quotes for string literals

Low

Single quotes are found in the above string variables, while double quotes are being used for other string literals.

```
require(msg.sender == owner, 'only admin');
require(msg.sender == owner, 'only admin');
```

Recomendation:

It is suggested to utilize double quotes for string literals.

Conformance to Solidity naming conventions

Low

Other functions besides constructors should use mixed case, such as getBalance, transfer, verifyOwner, addMember, and changeOwner.

```
timelock()
```

Recomendation:

We recommend renaming the function "timelock" to "timeLock" to follow the Solidity naming convention.

We are thankful for the opportunity to collaborate with the Kucoin Launchpad team.

This document's statements should not be taken as investment or legal advice, and the authors cannot be held responsible for any decisions made based on them.

It is suggested by the Tetragaze Security Team that the Kucoin Launchpad team implement a bug bounty program in order to encourage external parties to scrutinize the smart contract further.