La Cucina: LAC Token + Vault Security Analysis

by Pessimistic

This report is public

January 12, 2022

Abstract 2
Disclaimer 2
Summary 2
General recommendations 2
Project overview 3
Project description 3
Code base update 3
Code base update #2 3
Procedure 4
Manual analysis 5
Critical issues 5
Medium severity issues 5
Allocate extra funds (fixed) 5
Bug (fixed) 5
Tests issues (new) 5
Low severity issues 6
Code logic (fixed) 6
Message signing (fixed) 6
Project management (fixed) 6
Code quality 6
Notes 7
Overpowered roles 7

Abstract

In this report, we consider the security of smarts contracts of La Cucina project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, security audit is not an investment advice.

Summary

In this report, we considered the security of La Cucina smart contracts. We performed our audit according to the procedure described below.

The initial audit showed a few issues of medium severity including Allocate extra funds and a Bug. Also, several low-severity issues were found.

Some tests fail to pass. The documentation does not describe the off-chain part of the project. The operator role can censor users' transactions.

After the initial audit, the code base was updated. In this update, most of the issues were fixed.

Later the code base was updated again. In this update code quality has increased, no new issues were found.

General recommendations

We recommend addressing the remaining issues and providing the description for the off-chain mechanics of the project.

Project overview

Project description

For the audit, we were provided with La Cucina LAC Token project on a public GitHub repository, commit 8b0e5709dc4ac08b0dc930beb7abd5d6210547ce.

The project has README.md file with detailed documentation, the code base is thoroughly covered with NatSpec comments.

Three tests out of 59 do not pass, the code coverage is 82%.

The upgrade mechanism is out of scope of this review.

The total LOC of audited sources is 360.

Code base update

After the initial audit, the code base was updated. For the recheck, we were provided with commit 7671de1ca025fa7fd70410561cba3edb0f2b6256.

In this update, most of the issues were fixed, no new issues were found.

Code base update #2

After the first recheck, the code base was updated again. We were provided with commit 490309cc438e319bf295df749b84c05ddcc65925. This update introduces pausability to the **Vault** contract, adds new `view` function and includes new tests.

In this update code quality has increased, broken tests were fixed. However, some tests (mainly for new functions) do not pass.

Procedure

In our audit, we consider the following crucial features of the code:

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- Automated analysis
    - We scan project's code base with automated tool Slither.
    - We manually verify (reject or confirm) all the issues found by tools.
- Manual audit
    - We manually analyze code base for security vulnerabilities.
    - We assess overall project structure and quality.
- Report
    - We reflect all the gathered information in the report.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

The audit showed no critical issues.

Medium severity issues

Medium issues can influence project operation in current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

Allocate extra funds (fixed)

In **Vault** contract, `updateAllocatedFunds()` function can be called multiple times within a block by any user. This function makes a call to `getPendingAccumulatedFunds()` to calculate the amount to allocate. However, `getPendingAccumulatedFunds()` can only return a positive value. Therefore, the allocated amount can exceed expectations.

The issue has been fixed and is not present in the latest version of the code.

Bug (fixed)

In **Vault** contract, function `getPendingAccumulatedFunds()` does not assign a value to return for the case when `_isPeriodCompleted()` evaluates to `true` and `totalPeriodsCompleted > 0`.

The issue has been fixed and is not present in the latest version of the code.

Tests issues (new)

The project has tests. However, some tests do not pass. Testing is crucial for code security and audit does not replace tests in any way. We highly recommend covering the code base with tests and making sure that all tests pass and the code coverage is sufficient.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in the future versions of the code. We recommend taking them into account.

Code logic (fixed)

The `blockTime` variable is only used to calculate the block number from `block.timestamp` value. Consider using `block.number` instead to simplify code logic and optimize gas consumption.

The issue has been fixed and is not present in the latest version of the code.

Message signing (fixed)

Consider using on-chain `getVersionNumber` version value for domain separator instead of the one from back-end to eliminate the risk of double-versioning in **Vault** contract at line 85.

The issue has been fixed and is not present in the latest version of the code.

Project management (fixed)

- Package scripts require `truffle`, which is not declared as dependency.
- All the tools should be located in `devDependencies` section of **package.json** file. However, they are placed in `dependencies` section.

These issues have been fixed and are not present in the latest version of the code.

Code quality

- We recommend emitting events in **Vault** contract for better blockchain integration when `fundReceivers` values change.

  The issue has been fixed and is not present in the latest version of the code.

- In **Vault** contract, `_updateReleaseRate()` function is called inside `for`-loop. Consider precalculating required values and updating them with a single call.
- Consider using `return condition` syntax instead of `if (condition) {return true;} else {return false;}` to improve code readability, e.g. in `_isPeriodCompleted()` function of **Vault** contract.
- Consider minimizing the nested level of conditions to simplify code logic and improve readability in `getPendingAccumulatedFunds()` function of **Vault** contract.

- In `getPendingAccumulatedFunds()` function of **Vault** contract, the code blocks for `accumulatedFunds` value calculation is repeated four times with slight modifications. Consider using a universal formula instead and only modifying a single value in it depending on conditions.
- Most of the functions in the code are declared as `virtual`. Consider using this keyword only on case when the function is intended to be overridden.

Notes

Overpowered roles

The admin role can modify key parameters of the project.

> *Comment from developer: The admin roles will be assigned to a multi-signature wallet in order to avoid having a single EOW administrating the Vault.*
>
> *Governance voting - upon launch we are introducing a non-binding voting mechanic allowing our users to submit propositions, and vote on other user's proposals regarding future changes to the platform.*
>
> *In the future, this module might be upgraded to a full governance model, hence handing over the governance of some aspects of the platform, to the community.*

The operator role signs all the transactions from users and thus, can censor certain transactions.

> *Comment from developer: The operator/safe keeper is responsible for allowing wallet addresses claiming tokens from the Vault and this is a system design decision.*
>
> *The private keys of all the operators/safe keepers are stored securely.*

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Daria Korepanova, Security Engineer
Vladimir Tarasov, Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

January 12, 2022