Sahara Vesting Security Analysis

by Pessimistic

This report is public

April 22, 2022

Abstract

In this report, we consider the security of smart contracts of Sahara 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. A single 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, a security audit is not investment advice.

Summary

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

The audit showed several issues of medium severity, including ERC20 standard violation, Overpowered owner, Tests issues, and Discrepancy with documentation. Also, several low-severity issues were found.

After the initial audit, work was done to update the codebase. In this update, all detected issues were fixed or acknowledged.

General recommendations

We recommend implementing CI to run tests, calculate code coverage, and analyze code with linters and security tools.

Project overview

Project description

For the audit, we were provided with Sahara Vesting project on a public GitHub repository, commit 69127c87d6198b33dcc244942405b1f487ab4f08.

The scope of the audit included only **Vesting.sol** file.

The documentation for the project includes Sahara High Level Documentation.pdf and README.md files in the repository. The codebase contains detailed NatSpec comments.

The project does not compile. All 37 tests fail.

The total LOC of audited sources is 388.

Codebase update

After the initial audit, the codebase was updated. For the recheck, we were provided with commit f13a33657044fa863196d87fa50649bec2955361. In this update, all of the issues were fixed. A few new functions were added, no new issues were found.

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 the project's codebase with the automated tool Slither.
    - We manually verify (reject or confirm) all the issues found by the tool.
- Manual audit
    - We manually analyze the codebase for security vulnerabilities.
    - We assess the 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 the 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.

M01. ERC20 standard violation (fixed)

ERC-20 standard states:

```
Callers MUST handle false from returns (bool success). Callers MUST
NOT assume that false is never returned!
```

However, the returned value from `transferFrom` call is not checked at line 353.

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

M02. Overpowered owner (acknowledged)

The owner of the contract can:

1. Transfer ownership to any other address, including `0x0` address.
2. Add new vesting pools.
3. Add new beneficiaries to vesting pools.
4. Remove beneficiaries at any moment. Removed beneficiaries lose their tokens without any compensation. This functionality also allows the owner to front-run users who attempt to claim their tokens.
5. Change listing date for all pools.

In the current implementation, the system depends heavily on the owner of the contract. Thus, there are scenarios that can lead to undesirable consequences for the project and its users, e.g., if the owner's private keys become compromised. We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

> *Comment from the developers: Will use ownable upgradeable package which ensures that sending ownership to address 0x0 will be only in renounceOwnership function. Will use multisignature account for managing smart contract.*

M03. Tests issues (fixed)

The project has 37 tests. However, all these tests fail. Testing is crucial for code security. An audit does not replace tests in any way. We highly recommend covering the codebase with tests and ensuring that all tests pass and the code coverage is sufficient.

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

M04. Discrepancy with documentation (fixed)

1. According to the documentation, vesting for Marketing round should be `Linear (daily) over 24 months`. However, in the code, vesting period is set to `20` months at line 78.

2. According to the documentation, vesting for Advisors includes `39,000,000` tokens. However, in the code, the amount of tokens is `39,500,000` at line 81.

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

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Code quality (fixed)

Function `addVestingPool` assigns `vestingDurationInDays` and `vestingDurationInMonths` properties of `p` variable at lines 227–228. However, the function does not verify new values to be greater than `0`. The `unlockedTokenAmount` function use these values as a denominator at line 382. Therefore, division by zero can occur.

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

L02. Code quality (acknowledged)

The `transferOwnership` does not verify that new value is not a zero address.

> *Comment from the developers: Will use ownable upgradeable package which ensures that sending ownership to address 0x0 will be only in `renounceOwnership` function.*

L03. Code quality (fixed)

Functions `addToBeneficiariesList`, `removeBeneficiary`, and `changeListingDate` perform important changes that can affect all users of the project. Consider emitting proper events in these functions to keep track of such changes.

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

L04. Code quality (fixed)

The `Beneficiary` struct has `isWhitelisted` field declared at line 26. This field is only checked at line 162 within `onlyWhitelisted` modifier. However, the NatSpec comment for this modifier states:

```
Checks whether the address is beneficiary of the pool.
```

Since the only purpose of `isWhitelisted` field is to check whether the address is a beneficiary of the pool then:

1. Its name is misleading.
2. This field is redundant as testing for `totalTokens > 0` will give the same result.

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

L05. Non-retrievable tokens (fixed)

The contract does not have any functionality to retrieve tokens from it. Therefore, in the following situations tokens will be stuck on the contract:

- If users send tokens to the contract by mistake.
- If the amount of transferred tokens to the pool is greater than the vesting amount.
- If the owner deletes beneficiaries from the pool, tokens from these beneficiaries remain on the contract.

We recommend implementing functionality to retrieve tokens from the contract.

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

L06. Gas consumption (acknowledged)

The contract declares `Beneficiary` and `Pool` structures at lines 25–34 and 36–56 respectively. These structures have fields of `uint` type (which is alias for `uint256`). However, most of these values are small integers that can be packed into fewer storage slots. This allows decreasing gas consumption significantly.

> *Comment from the developers: There are some values in `Pool` structure that can be optimized, however, after gas cost comparison the decision was made to leave original `uint256` values to prevent accidental value overflow since the difference (from beneficiary perspective) is low on Polygon network.*

L07. Project management (fixed)

1. The project does not compile.
2. The project has no dependency management.
3. The repository does not contain configuration files and environment for running tests.

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

This analysis was performed by Pessimistic:

Vladimir Tarasov, Security Engineer
Vladimir Pomogalov, Security Engineer
Nikita Kirillov, Junior Security Engineer
Ivan Gladkikh, Junior Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

April 22, 2022