StakeWise Security Analysis

by Pessimistic

This report is public

March 1, 2022

Abstract

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

The initial audit showed a few issues of low severity. The overall code quality is good. The project has a detailed documentation.

After the initial audit the code base was updated, the code quality issue was fixed.

General recommendations

We recommend addressing the remaining issues.

Project overview

Project description

For the audit, we were provided with StakeWise project on a public GitHub repository, commit 5e43ba4b820676dea0147fd2e212ff8658ae8c06.

The project has public documentation at https://docs.stakewise.io/. Also, additional private documentation and explanations were provided for the audit. The codebase is covered with NatSpec comments.

All 204 tests pass.

The total LOC of audited sources is 1567.

Code base update

After the initial audit, the codebase was updated. For the recheck, we were provided with commit 27d11f3d1b50bcaec66e60ec0df332c561523d44.

In this update, the code quality issue was fixed, no 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 tools: Slither and SmartCheck.
  - We manually verify (reject or confirm) all the issues found by the tools.
- 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.

The audit showed no issues of medium severity.

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.

ERC20 standard issue

The ERC-20 standard states:

```
A token contract which creates new tokens SHOULD trigger a Transfer
event with the _from address set to 0x0 when tokens are created.
```

However, the `updateTotalRewards` function of **RewardToken** contract does not emit a `Transfer` event when creating new tokens.

> *Comment from the developers: We decided to leave this as it is.*

Gas consumption

`submitRewards`, `submitMerkleRoot`, and `registerValidator` functions of **Oracles** contract validate that supplied signatures belong to different oracles. This check iterates through nested loops and has a computational complexity of `O(n^2)`. We recommend sorting `signatures` array by `signer` address off-chain. In this case, the contract only needs to verify on-chain that each recovered `signer` address is greater than the previous one.

> *Comment from the developers: We decided to leave this as it is.*

Code quality (fixed)

In **RewardToken** contract, consider declaring functions `updateRewardCheckpoint` and `updateRewardCheckpoints` as `external` instead of `public` to improve code readability and optimize gas consumption.

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

Refactoring

In **RewardToken** contract, consider moving the check for null address at lines 141 and 187 to `balanceOf` function to minimize code duplication.

> *Comment from the developers: We decided to leave this as it is.*

Notes

Overpowered roles management

The project relies heavily on Oracle roles since a consensus of oracles determines the behavior of the system.

Admin roles can:

- Assign or remove roles. The admin of **Oracles** contract can assign and remove Oracles. Therefore, this role can manipulate the behavior of the system.
- Change fees in **RewardToken** contract.

Pauser roles can pause contracts.

All the roles are managed through the Gnosis Safe multisig with the Zodiac plugin, which minimizes the risk of compromising them.

This analysis was performed by Pessimistic:

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

March 1, 2022