Avata Staking

smart contracts
final audit report

July 2022

hashex.org

contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the Avata team to perform an audit of their smart contract. The audit was conducted between 28/03/2022 and 31/03/2022.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code is available at the @AVATA-Network/avata-contracts GitHub repository and was audited after the commit 23a8bfe.

The audited contracts: DistributionV2.sol, AllocationStaking.sol.

The audited contracts are deployed to the Avalanche C-Chain using the proxy:

DistributionV2

proxy - 0x54f59bdDc5bcE2b1e1dec6cE547F35ABdb7755e1

current implementation - 0x01183c756E02FAE0Fc4BfFaa8F3BACFd900ed08F

AllocationStaking

proxy - 0xA8cD6789Dabde5019b199093c15f7447e3C05578

current implementation - 0x981598bc172c3CC290964750a5c637328eeD3bb7

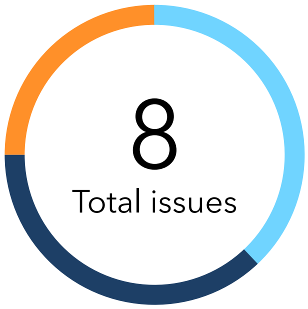
2.1 Summary

Project name	Avata Staking
URL	https://avata.network
Platform	Avalanche Network
Language	Solidity

2.2 Contracts

Name	Address
DistributionV2	0x54f59bdDc5bcE2b1e1dec6cE547F35ABdb7755e1
AllocationStaking	0xA8cD6789Dabde5019b199093c15f7447e3C05578

3. Found issues



●	High	2 (25%)
●	Low	3 (38%)
●	Info	3 (37%)

Ca1. DistributionV2

ID	Severity	Title	Status
Ca1Ib8	High	Unprotected mint by admin	Acknowledged
Ca1Ib7	Low	Lack of events	Acknowledged
Ca1Iba	Info	Incorrect documentation	Acknowledged

Ca2. AllocationStaking

ID	Severity	Title	Status
Ca2Ibb	High	Unable to withdraw funds	Acknowledged
Ca2Ibc	Low	No checks for initialize() paramaters	Acknowledged
Ca2Ib9	Low	Gas optimization	Acknowledged
Ca2Ibd	Info	Using safeTransfer	Acknowledged
Ca2Ibe	Info	Block gas limit	Acknowledged

4. Contracts

Ca1. DistributionV2

Overview

The contract stores the periods and the amount of emission of reward tokens. It also allows to mint reward tokens.

Issues

Ca1Ib8 Unprotected mint by admin ● High ✓ Acknowledged

The function `mintTokens()` allows the contract admin to mint tokens. And basically, this function is meant to be called by contract AllocationStaking.

But the contract owner has the ability to add (or change) another admin. In this case, the new admin is able to mint tokens. This can break tokenomics.

Recommendation

Restricting the ability to change the contract administrator.

Update

Since the contract was deployed with a proxy, its implementation can be changed at any time. In this case, there is also a risk that the new implementation will have unprotected minting on the part of the owner (or admin).

Ca1Ib7 Lack of events ● Low ✓ Acknowledged

The function `setPool()` doesn't emit events, which complicates the tracking of important off-chain changes.

Ca1Iba Incorrect documentation

Info — Acknowledged

a. Variable descriptions at L25 and L28 contain errors in the word 'precision'.

b. The documentation for the function `_calculatePoolAllocationReward()` does not match the code.

Ca2. AllocationStaking

Overview

The contract allows users to deposit tokens and receive rewards for this. The deposit can only be made for a certain period. The amount of rewards depends on the duration of such a period.

Withdrawals from the contract are only allowed during a certain time window after the end of the deposit period.

If no one has staked tokens during the reward distribution period, then the owner can take such unclaimed rewards using the `emergencyMint()` function.

Issues

Ca2Ibb Unable to withdraw funds

High — Acknowledged

User funds can be locked in the contract if the owner calls the `halt()` function of the contract because the function `withdraw()` would be inactive.

The situation could get worse if the owner is hacked. In this case, the funds may be blocked forever.

Recommendation

Restrict the owner's ability to halt the contract's operations.

Ca2Ibc No checks for initialize() paramaters

Low · Acknowledged

The input parameters of the `initialize()` function are not checked and cannot be updated later. This can lead to errors in the use of the contract if it was initialized with incorrect values.

Ca2Ib9 Gas optimization

Low · Acknowledged

a. The functions `add()`, `setDepositFee()`, `setAllocation()`, `setDistribution()`, `getiAVATAmount()`, `userStakesCount()`, `getUserStakes()`, `getUserStake()`, `getUserStakeIds()`, `deposited()`, `totalPending()`, `deposit()`, `withdraw()`, `collect()`, `restake()`, `compound()` can be declared as external to save gas.

b. There's no need to use the `_pid` argument in the `compound()` function, because only first pool is available for compounding.

Ca2Ibd Using safeTransfer

Info · Acknowledged

Since the contract uses the `SafeERC20Upgradeable` library, we recommend using `safeTranfer()` at L182, L582.

Ca2Ibe Block gas limit

Info · Acknowledged

The function `massUpdatePools()` iterates over an array of an unlimited size and may run out of block gas limit.

The owner of the contract should be aware of the issue when adding pools and must check gas usage.

5. Conclusion

2 high, 3 low, 3 informational severity issues were found.

The reviewed contracts are highly dependent on the owner’s account. Users using the project have to trust the owner and that the owner's account is properly secured.

This audit includes recommendations on improving the code and preventing potential attacks.

Appendix A. Issues severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Info.** Issues that do not impact the contract operation. Usually, info severity issues are related to code best practices, e.g. style guide.

Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

contact@hashex.org

@hashex_manager

blog.hashex.org

linkedin

github

twitter

HashEx

BLOCKCHAIN SECURITY