HashEx
BLOCKCHAIN SECURITY

Nazca Full

smart contracts final audit report

March 2024

🌐 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 Nazca team to perform an audit of their smart contracts. The audit was conducted between 04/03/2024 and 08/03/2024.

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 @NazcaMoney/contract Github repository after the fa2e0f9 commit.

2.1 Summary

Project name	Nazca Full
URL	https://nazca.money
Platform	Blast
Language	Solidity
Centralization level	● High
Centralization risk	● High

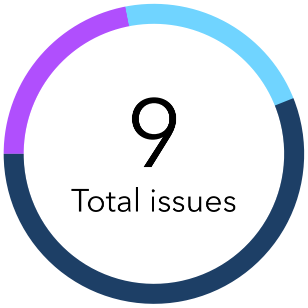
2.2 Contracts

Name	Address
NativeYieldDistributor	
CToken	
CErc20	
Comptroller	
CompoundLens	

3. Project centralization risks

The reviewed contracts is highly dependent on the owner’s account. The contracts are designed to be upgradeable meaning that accounts with privileged access can change implementations of the contracts. Users using the project have to trust the owner and that the owner's account is properly secured.

4. Found issues



●	Medium	2 (22%)
●	Low	2 (22%)
●	Info	5 (56%)

Cb6. NativeYieldDistributor

ID	Severity	Title	Status
Cb6If3	● Medium	Functions parameter validation	? Open
Cb6If5	● Low	Updating model parameters	? Open
Cb6If4	● Info	Lack of documentation (NatSpec)	? Open

Cb7. CToken

ID	Severity	Title	Status
Cb7If6	● Medium	Function parameter validation	? Open
Cb7If7	● Info	Lack of documentation (NatSpec)	? Open

Cb8. CErc20

ID	Severity	Title	Status
Cb8If8	● Info	Lack of documentation (NatSpec)	? Open

Cb9. Comptroller

ID	Severity	Title	Status
Cb9If9	● Info	Hardcoded liquidator address	? Open
Cb9Ifa	● Info	allBorrowers array length can't be zero	? Open

Cba. CompoundLens

ID	Severity	Title	Status
CbaIfb	● Low	Block gas limit	? Open

5. Contracts

Cb6. NativeYieldDistributor

Issues

Cb6If3 Functions parameter validation ● Medium ⓘ Open

1. The function `setReserveFactor()` allows setting the `reserveFactor` for the `market` in the range from 0 to 100%. If the `reserveFactor` is set to 100%, then all rewards will be allocated only to the `reserveAddress`. Consider the possibility of reducing the maximum limit for the `reserveFactor`.

2. The `setMarketParams()` function allows setting parameters for the market. We recommend adding validation for market parameters `params` to avoid setting incorrect parameters ().

Cb6If5 Updating model parameters ● Low ⓘ Open

We recommend updating the model (executing the `setModelParams()` function) together with updating the model parameters in a cToken. This will ensure the correct execution of the reward conditions described in the documentation.

Cb6If4 Lack of documentation (NatSpec) ● Info ⓘ Open

We recommend writing documentation using NatSpec Format. This would help in development, as well as simplify user interaction with the contract (including using the block explorer).

Cb7. CToken

Issues

Cb7If6 Function parameter validation ● Medium ⓘ Open

The `_setProtocolSeizeShare()` function allows changing the value of the state variable `protocolSeizeShareMantissa`. When the value exceeds 50%, liquidating positions may become disadvantageous for the liquidator, which could pose a risk to the entire project.

We recommend adding validation for the `newProtocolSeizeShareMantissa` parameter.

Cb7If7 Lack of documentation (NatSpec) ● Info ⓘ Open

We recommend adding NatSpec documentation for the new functions `claimGas()`, `claimYield()`, `updateRewards()`.

This would help in development, as well as simplify user interaction with the contract (including using the block explorer).

Cb8. CErc20

Issues

Cb8If8 Lack of documentation (NatSpec) ● Info ⓘ Open

We recommend adding NatSpec documentation for the new functions `getClaimableYield()`, `claimYield()`.

This would help in development, as well as simplify user interaction with the contract (including using the block explorer).

Cb9. Comptroller

Issues

Cb9If9 Hardcoded liquidator address ● Info ⓘ Open

In the `seizeAllowed()` function, a hardcoded address for the liquidator is used. We recommend using a variable for this address.

Cb9Ifa allBorrowers array length can't be zero ● Info ⓘ Open

The value of `borrowerIndexes` for the very first borrower in the project will always be equal to 0. Due to the check at L231, their address will never be removed from the `allBorrowers` array.

Cba. CompoundLens

Issues

CbaIfb Block gas limit ● Low ⓘ Open

The result of the external call `comptroller.getAllBorrowers()` is used in the functions `getAllBorrowersData()` and `getComptrollerData()`.

In case of a large number of borrowers in the project, these functions may fail due to the gas limit in the block. To resolve this issue, we recommend additionally implementing similar functions that will use the external call `comptroller.getBorrowers(startIndex, endIndex)`.

6. Conclusion

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.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. Issue status description

✓ **Resolved.** The issue has been completely fixed.

✓ **Partially fixed.** Parts of the issue have been fixed but the issue is not completely resolved.

✓ **Acknowledged.** The team has been notified of the issue, no action has been taken.

? **Open.** The issue remains unresolved.

Appendix C. 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

Appendix D. Centralization risks classification

Centralization level

- 🔴 **High.** The project owners can manipulate user's funds, lock user's funds on their will (reversible or irreversible), or maliciously update contracts parameters or bytecode.
- 🟠 **Medium.** The project owners can modify contract's parameters to break some functions of the project contract or contracts, but user's funds remain withdrawable.
- 🟢 **Low.** The contract is trustless or its governance functions are safe against a malicious owner.

Centralization risk

- 🔴 **High.** Lost ownership over the project contract or contracts may result in user's losses. Contract's ownership belongs to EOA or EOAs, and their security model is unknown or out of scope.
- 🟠 **Medium.** Contract's ownership is transferred to a contract with not industry-accepted parameters, or to a contract without an audit. Also includes EOA with a documented security model, which is out of scope.
- 🟢 **Low.** Contract's ownership is transferred to a well-known or audited contract with industry-accepted parameters.

✉ contact@hashex.org

@hashex_manager

blog.hashex.org

linkedin

github

twitter

HashEx

BLOCKCHAIN SECURITY