#HashEx
BLOCKCHAIN SECURITY

Nazca

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. The Nazca project is a fork of Compound V2 protocol with several audits available. The main differences are introduced in CToken markets and new NativeYieldDistributor contract in order to provide accumulation and distribution of the native Blast rewards for supported tokens.

The scope of the audit includes only the CToken and NativeYieldDistributor contracts.

Update. The Nazca team has responded to this report. The updated contracts are available in the same repository after the e7db4e4 commit and in the Blast chain at 0xB4225915598008B1f4790aF18A3a105BC188CFc4 (NativeYieldDistributor), 0x0A8fD60D354128cC5030c5719de79c732a7AEb0C and 0x07c7dcD6761c8b5F315A9B4431A980E8BD771240 (CToken). Both deployed versions of CTokens are immutable and can't be upgraded.

2.1 Summary

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

2.2 Contracts

Name	Address
NativeYieldDistributor	0xB4225915598008B1f4790aF18A3a105BC188CFc4
CToken	0x07c7dcD6761c8b5F315A9B4431A980E8BD771240

3. Project centralization risks

The reviewed contracts are highly dependent on the owner's account. The contracts are designed to be optionally 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.

The project owner also has the ability to set crucial system parameters beyond safe limits, possibly ruining the operability of the whole system or individual markets.

CbbCR13 Parameters validation

The contract's owner can change most of the system parameters without any validation of the input.

CbcCR14 Parameters validation

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.

Comments

Update

The updated contract ensures that the `protocolSeizeShareMantissa` parameter does not exceed 50%.

```
    function _setProtocolSeizeShare(uint newProtocolSeizeShareMantissa) external override
nonReentrant returns (uint) {
        ...
        if (newProtocolSeizeShareMantissa > protocolSeizeShareMaxMantissa) {
            revert SetProtocolSeizeSharerBoundsCheck();
```

```
        }
        ...
    }
```

4. Found issues

Medium	1 (17%)
Info	5 (83%)

Cbb. NativeYieldDistributor

ID	Severity	Title	Status
CbbI05	Medium	Limited test coverage	Resolved
CbbIfd	Info	Updating yield model parameters	Acknowledged
CbbIfe	Info	Lack of documentation (NatSpec)	Acknowledged
CbbI07	Info	Not validated external call	Acknowledged
CbbI06	Info	Delisted market rewards can't be claimed	Partially fixed

Cbc. CToken

ID	Severity	Title	Status
CbcI00	Info	Lack of documentation (NatSpec)	Acknowledged

5. Contracts

Cbb. NativeYieldDistributor

Overview

An additional contract to distribute native rewards from the Blast network according to single-kink interest model that extends Compound's interest model.

Designed to be deployed directly without proxy.

Issues

CbbI05 Limited test coverage

Medium Resolved

The project contains almost zero tests for the introduced functionality. There's also very scarce documentation for that functionality. This prevents us from fully verifying the correctness of the calculations of the distribution implemented by NativeYieldDistributor.

Recommendation

We urgently recommend increasing test coverage as well as remaining the original Compound tests. We also suggest enriching the documentation section, mentioning all used parameters and the rate calculation routines.

CbbIfd Updating yield model parameters

Info Acknowledged

Updating the parameters of the yield model (by executing the `setModelParams()` function) may not occur simultaneously with updating the parameters of the interest rate model of the cToken. For this reason, the graphs specified in the documentation may not fully coincide with the current parameters of the contracts. For example, the starting point of the JumpZone may differ between the yield model and the interest rate model of the cToken.

Team response

This flexibility allows us to adapt and upgrade our yield strategies in future according to our platform's needs without altering the underlying interest rate model.

CbbIfe Lack of documentation (NatSpec)

● Info ⊘ Acknowledged

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).

CbbI07 Not validated external call

● Info ⊘ Acknowledged

The `claimReward()` function receives the `market` address as parameter and skips its validation. Then it performs an external call to that address in the `getClaimableReward()` function. This arbitrary call can be used for phishing or siphoning, for example, by increasing user's gas costs to use it later by the attacker.

```
    function claimReward(address market) external {
        claimReward(market, msg.sender);
    }

    function claimReward(address market_, address account) private {
        uint256 amount = getClaimableReward(market_, account);
        (...)
    }

    function getClaimableReward(address market, address account) public returns (uint256)
{
        IMarket(market).updateRewards(account);
        return accounts[market][account].rewards;
    }
```

Recommendation

Consider prevent arbitrary markets by checking `market.lastUpdateBlock` or other parameters from the storage.

CbbI06 Delisted market rewards can't be claimed

Info Partially fixed

Rewards from the de-listed markets can't be claimed and remain locked in the contract due to require statements in the `updateModel()`, `updateRewards()` functions.

```
    function updateModel(uint256 cash, uint256 borrows, uint256 reserves, uint256
borrowRate, uint256 supply, uint256 borrowAmount) external {
        require(comptroller.isMarketListed(msg.sender), 'only the cToken may call this
function.');

        updateBorrowRewardRate(cash, borrows, reserves, borrowRate);
        updateYield(cash, supply, borrowAmount);
    }


    function updateRewards(address account_, uint256 supply, uint256 borrows) external {
        require(comptroller.isMarketListed(msg.sender), "only the cToken may call this
function.");
        ...
    }
```

Recommendation

Consider allowing such claims by adding `if(!comptroller.isMarketListed(msg.sender)) return` into `updateModel()`, `updateRewards()` functions.

Team response

Upon further review of our protocol, we've determined that the risk associated with delisting a market does not exist within our framework.

Cbc. CToken

Overview

An ERC-20 standard token originally developed by Compound Finance, designed to be deployed with proxy. The Nazca introductions include external calls to the NativeYieldDistributor contract tracking CToken changes, i.e. total supply, total borrows, borrow index, user's balances and borrows. The CToken's workflow is not affected by these calls.

Issues

CbcI00 Lack of documentation (NatSpec)

Info Acknowledged

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).

6. Conclusion

1 medium severity issue was found during the audit. 1 medium issue was resolved in the update. The reviewed contracts are highly dependent on the owner's account. See the centralization risks chapter.

This audit includes recommendations on code improvement and the prevention of 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.
- **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