HashEx

Polycrystal Finance

smart contracts audit report

Prepared for:

polycrystal.finance

Authors: HashEx audit team

June 2021

Contents

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.

DISCLAIMER: 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, 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.

Introduction

HashEx was commissioned by the Polycrystal Finance team to perform an audit of their smart contracts. The audit was conducted between June 21 and June 25, 2021.

The code located in github repository @polycrystal/polycrystal-contracts was audited after the commit `3df7e80`. It must be noted that /libs/Multicall.sol contract was not the subject of the audit. Documentation can be found on kingofcrystals1111.gitbook.io. The same contracts are deployed to Polygon (MATIC):
`0x76bF0C28e604CC3fE9967c83b3C3F31c213cfE64` CrystalToken,
`0xeBCC84D2A73f0c9E23066089C6C24F4629Ef1e6d` MasterHealer,
`0x5BaDd6C71fFD0Da6E4C7D425797f130684D057dd` CrystalMine,
`0x12fC8F5Cfb609981C6F6D141f0fb0BCE0b990145` Timelock.

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 to understand 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.

Contracts overview

`CrystalToken.sol`

Implementation of ERC20 and BEP20 [1] token standards with the minting open for the owner and governance from Yam Finance [2] (fork of Compound Governance Alpha, audit available [3]).

`MasterHealer.sol`

Staking contract similar to MasterChef by SushiSwap [4] (which audit's available [5,6]) with minor modifications.

`CrystalMine.sol`

Farm contract to reinvest tokens into Staking contract.

`Timelock.sol`

Timelock contract similar to Timelock by Compound Finance [7] (audit available [8]) with minor modifications.

Found issues

ID	Title	Severity	Response
01	Timelock: short minimum delay	Medium	Responded
02	CrystalToken: mint() is open for owner	Low	Informed
03	MasterHealer: BONUS_MULTIPLIER isn’t used	Low	Informed
04	CrystalMine: excessive requirements in notContract()	Low	Informed
05	CrystalMine: Pause and Unpause events are duplicated	Low	Informed
06	CrystalMine: userInfo variables not used	Low	Informed
07	CrystalMine: isContract() function is duplicated	Low	Informed
08	General recommendations	Low	Informed

#01 Timelock: short minimum delay — Medium

`MINIMUM_DELAY` constant is set to only 6 hours which in our opinion is too small. Moreover, the first admin change doesn't need any delay, see L82. We recommend changing the minimum delay of the contract to at least 12 hours or even better to transfer ownership to a new timelock with hardcoded `MINIMUM_DELAY` to 12 hours.
Response from the polycrystal.finance team: we are keeping at 6 hours just like Pancakeswap and ApeSwap for efficiency's sake.

#02 CrystalToken: mint() is open for owner — Low

`mint()` function at L18 is open for the owner. In case of token redeployment, ownership should be transferred to the MasterHealer contract as soon as possible. Users should check token ownership before using the token. It must be noted that the owner of the audited deployed token is the MasterHealer contract, only MasterHealer can mint tokens and token ownership cannot be transferred.

#03 MasterHealer: BONUS_MULTIPLIER isn't used — Low

`BONUS_MULTIPLIER` constant is set to 1 and therefore is useless. We recommend removing an unused variable from the contract.

#04 CrystalMine: excessive requirements in notContract() — Low

`notContract()` modifier at L900 contains two different checks of caller address. The second one is stricter and overlaps the terms of the extcodesize checking.

#05 CrystalMine: Pause and Unpause events are duplicated — Low

`CrystalMine` contract contains `Pause()` and `Unpause()` events L864 which are emitted at the same time as inherited from Pausable contract `Paused()` and `Unpaused()` events L741-746.

#06 CrystalMine: userInfo variables not used — Low

`crystalAtLastUserAction` and `lastUserActionTime` parameters of userInfo structs aren't used. The approximate user balance could be calculated with existing view functions and `user.shares`.

#07 CrystalMine: isContract() function is duplicated — Low

`_isContract()` function at L1162 duplicates the `isContract()` function of Address library L436.

`#08` `General recommendations` `Low`

Pragma version should be fixed.
Function `nonDuplicated()` in Master healer contract compares to a boolean constant.
Functions `setFeeAddress(), dev()` in the MasterHealer contract lack zero address check.
Functions `add(), set(), .deposit(), withdraw(), emergencyWithdraw(), dev(), setFeeAddress(), updateEmissionRate()` in the MasterHealer contract and `mint()` in the CrystalToken contract should be declared external.

Conclusion

No high severity issues were found.

Audit includes recommendations on the code improving and preventing potential attacks.

The audited contracts are deployed to Polygon (MATIC):
`0x76bF0C28e604CC3fE9967c83b3C3F31c213cfE64` CrystalToken,
`0xeBCC84D2A73f0c9E23066089C6C24F4629Ef1e6d` MasterHealer,
`0x5BaDd6C71fFD0Da6E4C7D425797f130684D057dd` CrystalMine,
`0x12fC8F5Cfb609981C6F6D141f0fb0BCE0b990145` Timelock.

References

1. BEP-20 standard
2. YAMGovernance on github
3. Compound Alpha audit by OpenZeppelin
4. SushuSwap's MasterChef contract
5. SushiSwap audit by PeckShield
6. SushiSwap audit by Quantstamp
7. CompoundFinance's Timelock contract
8. Timelock audit by OpenZeppelin

Appendix A. Issues' severity classification

We consider an issue to be critical, if it may cause unlimited losses, or breaks the workflow of the contract, and could be easily triggered.

High severity issues may lead to limited losses or break interaction with users or other contracts under very specific conditions.

Medium severity issues do not cause the full loss of functionality but break the contract logic.

Low severity issues are typically nonoptimal code, unused variables, errors in messages. Usually, these issues do not need immediate reactions.

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