#HashEx
BLOCKCHAIN SECURITY

Yield Parrot

smart contracts final audit report

October 2021

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 Yield Parrot team to perform an audit of their smart contracts. The audit was conducted between July 25 and July 29, 2021.

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.

2.1 Summary

Project name	Yield Parrot
URL	https://yieldparrot.finance
Platform	Binance Smart Chain
Language	Solidity

2.2 Contracts

Name	Address
LORYToken	`0xcD5D75Dbe75449A9021B6C570a41959eB571C751`
MasterLory	`0x1bee93b82275F3F215411bE49F948F8568e5e103`

3. Found issues

High	2 (29%)
Medium	4 (57%)
Low	1 (14%)

C1. LORYToken

ID	Severity	Title	Status
C1-01	High	Delegation double spend attack	Acknowledged

C2. MasterLory

ID	Severity	Title	Status
C2-01	High	Emission is not capped	Acknowledged
C2-02	Medium	Pools can be added with the same strategy	Acknowledged
C2-03	Medium	Pool.want and pool.strat could be set discoherent	Acknowledged
C2-04	Medium	EmergencyWithdraw may fail	Acknowledged
C2-05	Medium	Return values are not checked	Acknowledged
C2-06	Low	Gas optimization	Acknowledged

4. Contracts

C1. LORYToken

Issues

C1-01 Delegation double spend attack

High Acknowledged

Delegation power is not transferred with token transfers. An attacker can receive any delegation power as he want.

C2. MasterLory

Issues

C2-01 Emission is not capped

High Acknowledged

Owner can set arbitrary big value for the NATIVEPerBlock value. In such case token will be devalued and users will actually loose their rewards.

Team response

As the masterchef is already deployed and the function already created we are developing a timelock contract for the masterchef that will limit the max NATIVEPerBlock value to ensure there is no risk for users

C2-02 Pools can be added with the same strategy

Medium Acknowledged

The function add() does not check if a pool with a same strategy has already been added. Adding different pools with the same strategy will result in wrong pool allocation calculations in the updatePool() function.

C2-03 Pool.want and pool.strat could be set discoherent

Medium Acknowledged

Pool.want token may be set different from the pool.strat one. In such a case pool functionaly won't work.

Team response

Users can only deposit tokens in vaults where Pool.want and the wantToken in the Strat have coherency

C2-04 EmergencyWithdraw may fail

Medium Acknowledged

The functions emergencyWithdraw() calls external contracts that are out of scope of the current audit. If the strategy contract fails, user's won't be able to withdraw their funds even with the emergencyWithdraw() function.

C2-05 Return values are not checked

Medium Acknowledged

L1700, L1668 return value from IStrategy.withdraw is not used. Looks like IStrategy returns actual amount of tokens to be withdrawn that can be less than the passed amount to withdraw. In such a case there will be a discrepancy between stored balances and actual MasterLory token balance.

C2-06 Gas optimization

Low Acknowledged

Gas can be saved by declaring functions as external. MasterLory.setNATIVEPerBlock, MasterLory.inCaseTokensGetStuck, MasterLory.emergencyWithdraw, MasterLory.withdrawAll, MasterLory.deposit, MasterLory.set, MasterLory.add, NATIVEToken.mint should be declared external.

MasterLory.startBlock, MasterLory.ownerNATIVEReward, MasterLory.NATIVE should be declared constant.

EnumerableSet is unused.

5. Conclusion

2 high and 4 medium severity issues were found. Team responses were added below issues description.

Audit includes recommendations on the code improving 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.
- **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. 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

 @hashexbot

 blog.hashex.org

 linkedin

 github

 twitter

#HashEx
BLOCKCHAIN SECURITY