#HashEx
BLOCKCHAIN SECURITY

RCM Labs Inception

smart contracts
preliminary audit report
for internal use only

December 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). HashEx has exclusive rights to publish the results of this audit on company's web and social sites.

2. Overview

HashEx was commissioned by the **RCM Labs Inception** team to perform an audit of their smart contract. The audit was conducted between **2022-12-13** and **2022-12-16**.

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 @redcatmultiverse/inception Github repository after the e50c082 commit.

Update. The RCM Labs has responded to this report, the updated code is located in the same repository after cabec63 commit.

2.1 Summary

Project name	RCM Labs Inception
URL	https://www.redcatmultiverse.io/
Platform	Ethereum
Language	Solidity

2.2 Contracts

Name	Address
BoolMap	
Inception	

3. Found issues

Low	3 (43%)
Info	4 (57%)

C1. BoolMap

ID	Severity	Title	Status
C1-01	Low	Gas optimizations	Resolved
C1-02	Info	Custom implementation	Acknowledged
C1-03	Info	Variable default visibility	Resolved

C2. Inception

ID	Severity	Title	Status
C2-01	Low	Gas optimizations	Partially fixed
C2-02	Low	Input data is not validated	Partially fixed
C2-03	Info	Lack of events	Partially fixed
C2-04	Info	Code style	Acknowledged

4. Contracts

C1. BoolMap

Overview

A bitmap contract for whitelisted private sale, where every bit in the initialized slot is reserved for a token and can only be flipped by authorized users.

Issues

C1-01 Gas optimizations

Low Resolved

1. The `totalWhitelistedAddresses` variable is read from storage multiple times in the `optimizeSlots()` function. This can be avoided by using a local variable and a single read.

C1-02 Custom implementation

Info Acknowledged

We recommend using well-audited and community-acclaimed implementations from the OpenZeppelin library where it's possible. The BitMap contract should be used instead of BoolMap.

C1-03 Variable default visibility

Info Resolved

The variable `mintMap` has default visibility. Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

C2. Inception

Overview

An ERC-721 standard contract inherited from the solmate implementation with the support of the EIP-2981 royalty standard (OpenZeppelin implementation). A single royalty of up to 100% can be set for all tokens.

The token has a fixed max total supply of 8888 tokens with 2 reserved parts for 2 rounds of private sales and a team reserve. Public sale rounds consist of a classic sale with a fixed price and a Dutch auction round with a step-reducing price.

Issues

C2-01 Gas optimizations

Low Partially fixed

1. The `DutchAuction` struct may use compact unit types for storing prices and other parameters.
2. The price variables can be packed together into one storage slot by casting to `uint80` type.
3. Since the arguments `to` and `amount` of the function `whitelistAddresses()` are read-only, they can be declared as `calldata` instead of `memory` to save gas.
4. Consider using the `<=` operator on L240 for the first if-statement of the `auctionPrice()` function.

C2-02 Input data is not validated

Low Partially fixed

Most of the governance functions and the constructor do not validate the input data by any means. This may lead, for example, to zero-price purchases or an inaccessible Dutch auction.

Also, consider validating the maximum value for a royalty amount in the `setRoyalty()` function.

C2-03 Lack of events

Info | Partially fixed

`onlyOwner` governance functions don't emit any events, which complicates the tracking of important off-chain changes.

C2-04 Code style

Info | Acknowledged

1. We recommend using an enum for the `saleState` variable instead of `uint`. This will increase the code readability.

2. We recommend defining default values for prices in the contract constructor to prevent cases when the contract owner forgot to set a price.

5. Conclusion

3 low severity issues were found during the audit. 1 low issue was resolved in the update.

The reviewed contract is 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 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.
- **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. 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

 contact@hashex.org

 @hashex_manager

 blog.hashex.org

 linkedin

 github

 twitter

#HashEx
BLOCKCHAIN SECURITY