#HashEx
BLOCKCHAIN SECURITY

CoinBurp BigTownChef

smart contracts final audit report

March 2022


hashex.org


contact@hashex.org

Contents

1. Disclaimer 3

2. Overview 4

3. Found issues 6

4. Contracts 8

5. Conclusion 12

Appendix A. Issues severity classification 13

Appendix B. List of examined issue types 14

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 **CoinBurp** team to perform an audit of their smart contract. The audit was conducted between 18/03/2022 and 22/03/2022.

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 the @coinburp/coinburp-contracts GitHub repository and was audited after the commit 1668127.

Update: the CoinBurp team has responded to this report. The updated code is located in the GitHub repository after commit 6c1abdd.

2.1 Summary

Project name	CoinBurp BigTownChef
URL	https://github.com/coinburp/coinburp-contracts/pull/81/commits/9c722f7e8e7590b3340903bee8b1ab4a2ea8a4f1
Platform	Ethereum
Language	Solidity

2.2 Contracts

Name	Address
ChefAvatar	
ChefSaleManager	
ChefRevealProvider	

3. Found issues

Medium	1 (7%)
Low	12 (80%)
Info	2 (13%)

C1. ChefAvatar

ID	Severity	Title	Status
C1-01	Medium	Change of ChefSaleManager contract	Acknowledged
C1-02	Low	No checks for constructor parameters	Resolved
C1-03	Low	Gas optimization	Partially fixed
C1-04	Low	Variable default visibility	Resolved
C1-05	Low	Lack of events	Resolved

C2. ChefSaleManager

ID	Severity	Title	Status
C2-01	Low	Integer underflow	Resolved
C2-02	Low	Function parameter validation	Partially fixed
C2-03	Low	Variable default visibility	Resolved

C2-04	Low	Lack of events	Resolved
C2-05	Low	Gas optimization	Resolved
C2-06	Info	Comment mismatch	Resolved
C2-07	Info	Lack of documentation	Resolved

C3. ChefRevealProvider

ID	Severity	Title	Status
C3-01	Low	Gas optimization	Resolved
C3-02	Low	Lack of events	Resolved
C3-03	Low	No checks for constructor paramaters	Acknowledged

4. Contracts

C1. ChefAvatar

Overview

The ERC721a contract allows buying NFTs through the ChefSaleManager contract.

It has a minting limit determined by the owner at the time of deployment.

Issues

C1-01 Change of ChefSaleManager contract

Medium Acknowledged

The owner has the ability to change the `ChefSaleManager` contract using setChefSaleManager. The new contract may implement the ability to mint new tokens without payment using the `mint()` function.

C1-02 No checks for constructor parameters

Low Resolved

The input constructor parameters are not checked and cannot be updated later. This can lead to errors in the use of the contract if it was initialized with wrong values.

C1-03 Gas optimization

Low Partially fixed

a. The state variable `_baseTokenURI` can be declared as immutable to save gas.

b. The functions `setChefRevealProvider()`, `setChefSaleManager()`, `exists()`, `tokenURI()`, `mint()` can be declared as external to save gas.

C1-04 Variable default visibility

Low Resolved

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

C1-05 Lack of events

Low Resolved

The functions `setChefRevealProvider()`, `setChefSaleManager()` don't emit events, which complicates the tracking of important off-chain changes.

C2. ChefSaleManager

Overview

The contract allows users to buy NFT tokens (ChefAvatar) on presale or public sale. The public sale can be at a fixed price or through a Dutch auction.

Issues

C2-01 Integer underflow

Low Resolved

The calculation at L110 can lead to integer underflow. This would block the ability to buy tokens. We recommend addressing this situation and making it possible to purchase at `dutchEndPrice` price.

C2-02 Function parameter validation

Low Partially fixed

The input parameters of contract constructor and functions `setPrices()`, `setPresaleConfig()`, `setPublicConfig()`, `setPublicSaleMaxPurchaseQuantity()`, `configureDutch()` are not checked. This can lead to errors in the use of the contract if the wrong values have been given.

C2-03 Variable default visibility

Low Resolved

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

C2-04 Lack of events

Low Resolved

The functions `setMerkleRoot()`, `setTreasury()`, `setPrices()`, `setPresaleConfig()`, `setPublicConfig()`, `setPublicSaleMaxPurchaseQuantity()`, `setPublicSalePricingModel()`, `configureDutch()` don't emit events, which complicates the tracking of important off-chain changes.

C2-05 Gas optimization

Low Resolved

The functions `setMerkleRoot()`, `setTreasury()`, `setPrices()`, `setPresaleConfig()`, `setPublicConfig()`, `setPublicSaleMaxPurchaseQuantity()`, `setPublicSalePricingModel()`, `configureDutch()`, `presaleBuy()`, `publicBuy()` can be declared as external to save gas.

C2-06 Comment mismatch

Info Resolved

a. The code comment at L17 does not match the value of the variable `presaleStart`.

b. The code comment at L19 does not match the value of the variable `publicStart`.

C2-07 Lack of documentation

Info Resolved

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).

C3. ChefRevealProvider

Overview

The contract helps to interact with the ChainLink random generator to generate the `tokenURI` in the contract ChefAvatar.

Issues

C3-01 Gas optimization

Low Resolved

State variables `keyHash`, `chefAvatar` can be declared as `immutable` to save gas.

C3-02 Lack of events

Low Resolved

The function `setFee()` doesn't emit events, which complicates the tracking of important off-chain changes.

C3-03 No checks for constructor paramaters

Low Acknowledged

The input constructor parameters are not checked and cannot be updated later. This can lead to errors in the use of the contract if it is initialized with wrong values.

5. Conclusion

1 medium, 12 low, 2 informational severity issues were found. 10 low and
1 informational severity issues were resolved in the update.

This audit includes recommendations on improving the code 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.
- **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. 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