HashEx

CoinBurp

smart contracts audit report

Prepared for:

coinburp.com

Authors: HashEx audit team

July 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 CoinBurp team to perform an audit of their smart contracts. The audit was conducted between July 05 and July 17, 2021.

The code located in github repository @coinburp/coinburp-contracts was audited after the commit 1421003. The code was provided without any documentation.

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.

Update: CoinBurp team has responded to this report. Individual responses were added after each item in the section. The updated code is located in the same repository after the 0bd1c5b commit.

Contracts overview

`BurpERC20Token.sol`

ERC20 standard [1] token with fixed supply.

`NFTRarityRegister.sol`

Contract for NFT registration.

`RaffleTicket.sol`

Mintable ERC1155 standard [2] token for Raffle.

`RaffleVRFConsumer.sol, VRFConsumerBase.sol & VRFRequestIDBase.sol`

Support contracts for using Chainlink VRF [3].

`RaffleAccessControl.sol`

Access control contract for Raffle.

`Raffle.sol`

Lottery contract randomized by Chainlink VRF.

`RewardStreamer.sol & RewardStreamerLib.sol`

Reward manager for Staking.

`TokenHelper.sol`

Support contract for extended token transfers.

`Staking.sol & StakingLib.sol`

Staking contract with NFT boost.

Found issues

ID	Title	Severity	Response
01	TokenHelper: fail of token transferFrom() is not handled	Critical	Fixed
02	TokenHelper: return statement checks for wrong address in transferFrom()	High	Fixed
03	TokenHelper: _mintTickets() does not check the result of the internal call	High	Fixed
04	RaffleAccessControl: misuse of OZ AccessControl	High	Fixed
05	RewardStreamerLib: rewardStreamsLength is not updated in addRewardStream()	Medium	Fixed
06	Staking: lock duration not limited from above	Medium	Responded
07	StakingLib: NFT transfers not checked	Medium	Responded
08	StakingLib: getStakerRewardFromCurrentPeriod() ignores excludeLast parameter	Medium	Fixed
09	Raffle: withdrawGracePeriod should be limited	Medium	Fixed
10	Raffle: draftWinners() callable only by the owner	Medium	Responded
11	RewardStreamerLib: _isContract() misuse	Medium	Fixed
12	RewardStreamerLib: getRewardAndUpdateCursor() is unsafe without description	Low	Fixed
13	RewardStreamerLib: addRewardStream() excessive computations	Low	Fixed
14	RewardStreamerLib: unsafeGetRewardsFromRange() and getRewardAndUpdateCursor() could exceed gas limit	Low	Responded
15	Raffle:inconsistent comment	Low	Fixed

16	StakingLib: _userStakes[] always grows in length	Low	Responded
17	General recommendations	Low	Fixed

#01 TokenHelper: fail of token transferFrom() is not handled — Critical

The function `transferFrom()` in TokensHelper is a wrapper of the ERC721 [4] transferFrom function. The wrapper uses a low-level `call()` function (L44) but does not check the result of the operation. If the transferFrom function of ERC721 token fails (for example, the transfer is called not from the token owner), the call function will not revert and will return false.

The `TokensHelper.transferFrom()` function is used in the StakingLib function `_addNftToStakeAndApplyMultiplier()` and transfers the NFT token from the caller to the contract. This function won't fail and will return a successful result if called by an arbitrary address but the NFT token was already successfully staked, i. e. its current owner is the Staking contract.

Recommendation: add a `require()` statement that checks the result of the `call()` function.

Update: the issue was fixed.

#02 TokenHelper: return statement checks for wrong address in transferFrom() — High

The function `transferFrom()` in TokensHelper checks that the owner of the token is the caller contract, but not the "`to`" address passed in the function parameter.

Recommendation: the return statement in L50 should check `abi.decode(data, (address)) == to`.

Update: the issue was fixed.

#03 TokenHelper: _mintTickets() does not check the result of the internal call — High

The result of the `call()` function in the L58 is not checked. If the mint function in a contract implementing IRaffleTicket fails, the `call()` will return false. We can't say if this is the desired behavior as there is no documentation on this function, so we treat this non-transparent behavior as a high severity issue.

Recommendation: add a `require()` statement checking the result of the IRaffleTicket.mint() call.

Update: the issue was fixed.

#04	RaffleAccessControl: misuse of OZ AccessControl	High

`addMinter()` function has the `onlyMinter` modifier and calls `_setupRole()` function that should be called only in the constructor [5]. The current implementation bypasses the MINTER_ROLE admin as any minter is able to add more minters.

Recommendation: use `grantRole()` function.

Update: the issue was fixed.

#05	RewardStreamerLib: rewardStreamsLength is not updated in addRewardStream()	Medium

The function `addRewardStream()` adds RewardStream to a RewardStreamInfo structure but does not update the `rewardStreamsLength` field in this structure. It must be noted that the `rewardStreamsLength` field is not used in the contracts.

Recommendation: we recommend removing the `rewardStreamsLength` field. If there is a need for getter `rewardStreams[]` length for frontend/backend services, we recommend adding a special getter function for it in the contract.

Update: the issue was fixed.

#06	Staking: lock duration not limited from above	Medium

`addLockDuration()` and `updateLocks()` functions (L75, 88) don't filter input parameters. Incorrect interpretation of duration, e. g. milliseconds or timestamp of (local time + duration), may result in the user's lock for years.

Recommendation: add the requirements for min-max values of input parameters in every set function even if it's onlyOwner.

CoinBurp team response: the numbers are not time, but blocks number. So it should be safe enough.

Comment on response: even if the input parameter is block number, it could be miscalculated from wrong timestamps.

#07	StakingLib: NFT transfers not checked	Medium

TokenHelper implements a wrapper over the ERC721 [4] transferFrom() function and returns the transfer result. However, `removeNftFromStake()` function in StakingLib contract has no check of that result (L790).

Recommendation: current realization of TokenHelper contains the #02 issue that should be fixed. After that, we recommend checking the transfer result either in the wrapper function or after each call to it.

CoinBurp team response: removeNftFromStake() function should fail silently by design, see L825.

Comment on response: if the NFT transfer fails inside `unstake()` function, users should use `unstakeERC721()`.

#08	StakingLib: getStakerRewardFromCurrentPeriod() ignores excludeLast parameter	Medium

`getStakerRewardFromCurrentPeriod()` function handles it's `excludeLast` parameters as always true.

Recommendation: update the description or implement handling for `excludeLast == false`.

Update: the issue was fixed.

#09	Raffle: withdrawGracePeriod should be limited	Medium

changeWithdrawGracePeriod() function L170 should have a limiter for the minimum of `withdrawGracePeriod` parameter to prevent malicious owner from `unlockUnclaimedPrize()` immediately.

Recommendation: add the requirements for min-max values of input parameters in every set function even if it's onlyOwner.

Update: the issue was fixed.

#10	Raffle: draftWinners() callable only by the owner	Medium

`draftWinners()` function L214 is callable only by the owner which makes it possible for a malicious owner to wait for the `withdrawGracePeriod` and return the unclaimed prize.

Recommendation: open `draftWinners()` for public use.

CoinBurp team response: that's a fair point we have been thinking about. The alternative was to leave this open to anyone to provide an entropy value, but by leaving it open, a Chainlink operator could call this function with an entropy number favourable to them.

In both cases, a bad actor could influence the game, but the chances that it will be the owner (meaning Coiburp itself) are extremely slim and easy to spot. While a Chainlink operator using a favourable entropy will just be impossible to spot.

#11	RewardStreamerLib: _isContract() misuse	Medium

`_isContract()` function is a copy of `isContract()` from the Address library by OpenZeppelin. It safely returns 'true' but one should not rely on 'false' returning value as contracts in construction have 0 code size. However, this function is used in `unsafeGetRewardsFromRange()` view function which is safe but questionable.

Recommendation: use `tx.origin == msg.sender` to check `isNotContract`.

Update: the issue was fixed.

#12	RewardStreamerLib: getRewardAndUpdateCursor() is unsafe without description	Low

`getRewardAndUpdateCursor()` function could skip reward periods if fromBlock is high enough. Current implementation of functions using `getRewardAndUpdateCursor()` is safe.

Update: the issue was fixed.

#13	RewardStreamerLib: addRewardStream() excessive computations	Low

`addRewardStream()` function reads the same variable twice L48, 52. L52 contains unnecessary checked addition, requiring less or equal should be enough.

Update: the issue was fixed.

#14 RewardStreamerLib: unsafeGetRewardsFromRange() and getRewardAndUpdateCursor() could exceed gas limit — Low

`for()` loop over uint256 length in L103, 190 could exceed block gas limit and break all the depending functions. `unsafeGetRewardsFromRange()` is used only in view functions and could be optimized with reducing the reads (e.g. in `for()` conditions), limited use of unchecked math and translating parameters through `iterateRewards()` cycle. `getRewardAndUpdateCursor()` needs huge `rewardStreams.length` to exceed the gas limit, but is used in crucial functions like `addNftToStake()` and `unstake()`.

CoinBurp team response: we need to make a distinction between the two functions. The first unsafeGetRewardsFromRange should not be used by any smart contract (in fact has a check against that). The second, getRewardAndUpdateCursor, uses a cursor exactly to avoid big loops. With the cursor, we can store the last index checked and when calling it again the result will be almost always O(1) or O(2) in the case when we pass from one RewardStream to another, and only for one staker (again, because the cursor will be stored and the difficulty will be again O(1)). Another thing to keep in mind is that the change of period happens 1 a year, so for example a loop of 7 iterations would mean that no one has ever staked or unstaked for 7 years straight.

#15 Raffle:inconsistent comment — Low

L232 comment should mention the situation of a raffle conclusion without prizes.

Update: the issue was fixed.

#16 StakingLib: _userStakes[] always grows in length — Low

`unstake()` function uses `_resetStake()` method L764 which doesn't pop the corresponding element of UserStake array.

CoinBurp team response: this is by design, as we keep track of stake indexes and we need to guarantee a stakeIndex is always unique and never reused.

#17 General recommendations Low

RewardStreamer: typos in L26, 47.

RewardStreamerLib: typos in L33, 121.

Staking: typos in L42, 62, 110, 227, 294, 370, 375, 403.

StakingLib: typos in L46, 67, 290, 87, 129, 147, 179, 572, 637, 213, 234, 295, 715, 777.

IRaffle: typo in L45.

IRaffleTicket: typo in L7.

RaffleTicket: typo in L10.

Raffle: typos in L157, L178, L268, L280.

SafeMath library could be removed with the 0.8.0+ compiler.

We recommend avoiding `require()` statements in view functions if possible.

Update: the issues were partially fixed.

CoinBurp team response: for some of the methods, a require is crucial, as returning a default value (e.g. 0) can still be a valid result that should not be returned.

Conclusion

1 critical and 3 high severity issues were found. The contracts are 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.

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

Update: CoinBurp team has responded to this report. Most of the issues were fixed including all the high and critical ones. Updated contracts are located in the same repository after the 0bd1c5b commit.

References

1. EIP-20: ERC-20 Token Standard
2. EIP-1155: ERC-1155 Multi Token Standard
3. Introduction to Chainlink VRF
4. EIP-721: ERC-721 Non-Fungible Token Standard
5. AccessControl.sol by OpenZeppelin

Appendix A. Issues' severity classification

We consider an issue 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