HashEx
Blockchain Security

LiquidCollectibles

smart contracts final audit report

November 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 Liquid Collectibles team to perform an audit of their smart contracts. The audit was conducted between October 24 and November 3, 2021.

The code located in the GitHub repository @liquidcollectibles/liquidcollectibles-contracts was audited after the commit 32d911a. Only 4 contracts were in the scope of the audit. The updated code was re-checked after the 7fcc923 commit.

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.

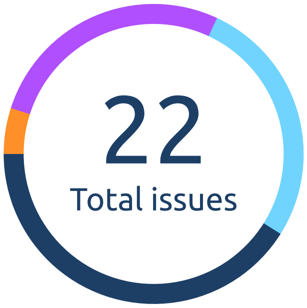
2.1 Summary

Project name	LiquidCollectibles
URL	https://liquidcollectibles.io
Platform	Binance Smart Chain
Language	Solidity

2.2 Contracts

Name	Address
LicoToken	0x4F3266a56589357B4f8082918b14B923693e57f0
MasterChefV3	0x25eF2439ee92552B6f58463394E66D289F9e7b7C
LicoVault	0x6c3B1Ff7481fE0B106D046CfE60c5751caA30c7d
RewardTimer	0xd05176C5A049C0ACF946D8992016dE4c9A9761a1

3. Found issues



■	High	1 (5%)
■	Medium	6 (27%)
■	Low	6 (27%)
■	Informational	9 (41%)

LicoToken

ID	Title	Severity	Status
01	Minting is open for the owner	■ Medium	Resolved
02	Typo in constructor parameter name	■ Informational	Resolved

MasterChefV3

ID	Title	Severity	Status
01	External calls to rewarders should be wrapped in try/catch	■ High	Resolved
02	massUpdatePools() gas limit	■ Medium	Acknowledged
03	setStartTime() can be set in future	■ Medium	Resolved
04	add/set functions are called		

	with optional massUpdatePools()	■ Low	Resolved
05	Gas optimizations	■ Low	Partially fixed
06	Input parameters not filtered	■ Low	Acknowledged
07	pendingLico() uses balance	■ Informational	Resolved
08	Redundant code	■ Informational	Acknowledged
09	Inconsistent comments	■ Informational	Resolved

LicoVault

ID	Title	Severity	Status
01	Admin functions don't emit specific events	■ Low	Acknowledged
02	Non standard ERC20 tokens aren't supported	■ Informational	Acknowledged
03	_isContract() function is unreliable	■ Informational	Acknowledged
04	Constructor parameters aren't checked for input values	■ Informational	Acknowledged

RewardTimer

ID	Title	Severity	Status
01	Unlimited rewardPerSecond from owner	■ Medium	Acknowledged
02	Function set() does not update pool	■ Medium	Acknowledged
03	Possible discrepancy in user shares calculation	■ Medium	Acknowledged
04	Rewards pool must be replenished manually	■ Low	Acknowledged
05	Gas optimizations	■ Low	Acknowledged
06	Ownable contract version	■ Informational	Acknowledged
07	Inconsistent comments & typos	■ Informational	Acknowledged

4. Contracts

4.1 LicoToken

4.1.1 Overview

A token mintable by a single owner with a governance model initially introduced by Compound Finance and now available in OpenZeppelin library [1].

4.1.2 Issues

01. Minting is open for the owner

■ Medium ✓ Resolved

This token is meant to be used with a MasterChef-like contract. Whenever this code is deployed, users should pay attention if the ownership is transferred to the contract that can't mint the token uncontrollably and can't transfer the token's ownership further.

Update

The ownership was transferred to the MasterChefV3 contract.

02. Typo in constructor parameter name

■ Informational ✓ Resolved

There is a typo in the initiaSupply parameter in the constructor.

4.2 MasterChefV3

4.2.1 Overview

A staking contract similar to MasterChef by SushiSwap. Takes a limited percent fee (up to 10%) with every deposit. Mints LICO rewards to the stakers and the contract owners.

4.2.2 Issues

01. External calls to rewarders should be wrapped in try/catch

▪ High ✓ Resolved

External calls in deposit(), withdraw() and externalWithdraw() functions may fail and cause a lock of the users' funds.

Recommendation

Wrap these calls in try/catch method with additional error logging in the catch section.

02. massUpdatePools() gas limit

▪ Medium ⓘ Acknowledged

for() loop in massUpdatePools() over the PoolInfo[] array length may cause the block gas limit exceedance of updateEmissionRate() calls.

Recommendation

The owners should monitor average gas costs and limit the number of pools. Keep in mind that block gas limit could be decreased in future network updates.

03. setStartTime() can be set in future

▪ Medium ✓ Resolved

Calling the setStartTime() function with the timestamp in distant future may lead to malfunction of the pools that would be added with that startTime. Also, the function lacks a corresponding event.

Recommendation

Limit the input value of the setStartTime() function.

04. add/set functions are called with optional massUpdatePools()

▪ Low ✓ Resolved

add() and set() functions have an optional _withUpdate flag which calls massUpdatePools() if set true, and may cause unfair rewards in case of rarely updated pools [2].

05. Gas optimizations

- Low ◎ Partially fixed

PoolInfo structure could be rearranged to store address with the uint16 into single 256-bit slot.

The updatePool() function should prematurely return in case of zero totalLico value in L191 when the cap is reached and the emission rate is set to 0.

06. Input parameters not filtered

- Low ⓘ Acknowledged

Input parameters of constructor, dev() and setFeeAddress() functions aren't checked for zeros and/or unreasonably high values.

07. pendingLico() uses balance

- Informational ✓ Resolved

The pendingLico() function uses pool.stakeToken.balanceOf(address(this)) instead of pool.totalStaked.

08. Redundant code

- Informational ⓘ Acknowledged

The pool.totalStaked variable introduced simultaneously with poolExistence[]. The first one allows using pools with the same stakeToken address. The second one is used to track and deny these duplicating pools. Also, poolExistence[] mapping could be transformed into (token address)=>(pid+1), providing a helpful public getter.

09. Inconsistent comments

- Informational ✓ Resolved

Inconsistent comment L43.

TODO comment in L193.

4.3 LicoVault

4.3.1 Overview

Vault contract for the MasterChefV3. Tracks the users' shares and collects fees on every harvest() call and early withdrawal in less than 3 days.

4.3.2 Issues

01. Admin functions don't emit specific events

- Low ⓘ Acknowledged

setAdmin(), setTreasury(), setPerformanceFee(), setCallFee(), setWithdrawFee(), setWithdrawFeePeriod() and emergencyWithdraw() functions L155-212 don't emit any specific events. On the other hand, pause() and unpause() functions L230-239 emit duplicated events Pause/Paused & Unpause/Unpaused.

02. Non standard ERC20 tokens aren't supported

- Informational ⓘ Acknowledged

The contract is designed (and deployed) to work only with classic ERC20 tokens (i.g. LICO). Transfer-taxed tokens or custom tokens with internal hooks are not supported.

03. _isContract() function is unreliable

- Informational ⓘ Acknowledged

notContract() modifier contains an unreliable condition in L87: _isContract() relies on extcodesize and should not be used to check the 'address is not a contract' statements [3].

04. Constructor parameters aren't checked for input values

- Informational ⓘ Acknowledged

Deploying the contract with zero addresses could cause silent losses of the fees if the treasury address was set to 0.

4.4 RewardTimer

4.4.1 Overview

Optional secondary rewards for the pool of MasterChefV3. This contract is called by the MasterChefV3 with every user balance update.

4.4.2 Issues

01. Unlimited rewardPerSecond from owner

▪ Medium (!) Acknowledged

The setRewardPerSecond() function hasn't the upper limit for the input value of rewardPerSecond. The owner has the ability to drain the rewards pool completely in a single transaction.

02. Function set() does not update pool

▪ Medium (!) Acknowledged

The function() set changes allocation pool, but does not call updatePool() prior to it. It may lead to a situation when pending rewards are recalculated after the set() function is called.

Recommendation

Update pool in the set() function:

```
    function set(uint256 _pid, uint256 _allocPoint) external onlyOwner {
        updatePool(_pid);
        totalAllocPoint =
totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
        poolInfo[_pid].allocPoint = _allocPoint;
        emit LogSetPool(_pid, _allocPoint);
    }
```

03. Possible discrepancy in user shares calculation

■ Medium ⓘ Acknowledged

The RewardTimer contract is supposed to update the user's balance after the user deposits or withdraws tokens in the MasterChefV3 contract.

The function onReward() in the RewarderTimer contract may not be called when balances are changed in the MasterChefV3 contract. This may lead to errors in the rewards calculation.

If the RewardTimer is set in the MasterChefV3 contract after the first users make their deposits, the rewards aren't calculated for them until a user deposits or withdraws tokens.

Let's see another situation when rewardTimer is initially set in the MasterChefV3 contract and then is temporarily turned off. A user has a non-zero amount in the RewardTimer contract. After the RewardTimer was unset in the MasterChefV3 contract users withdraw these tokens. Then the RewardTimer is set back, the user deposits some tokens and gets a reward in the RewardTimer contract as if he held his initial balance all the time.

Recommendation

As the RewardTimer contract is already deployed, we recommend adding new pools with RewardTimer or not adding RewardTimer in the current version to pool at all. Also, we recommend not setting back a RewardTimer if it has been switched off.

04. Rewards pool must be replenished manually

- Low ⓘ Acknowledged

The reward token balance of this contract must be replenished externally. getSecondsToEmptyRewards() function returns (totalRewardsAllocated - totalRewardsPaid) / rewardPerSecond without checking the current balance. Users should check the available balance before claiming their rewards and postpone the claiming if possible.

05. Gas optimizations

- Low ⓘ Acknowledged

Multiple reads from the storage: balance in L79-80, user.amount in L74-82.

06. Ownable contract version

- Informational ⓘ Acknowledged

Standard Ownable contract by OpenZeppelin isn't designed to work with initializable contracts. There's a special version in the contracts-upgradeable repository with the __Ownable_init() function to be called in the initializer of the contract.

07. Inconsistent comments & typos

- Informational ⓘ Acknowledged

Inconsistent copy-paste comment L53. Typos 'balace' and 'conract' in L211.

5. Conclusion

1 high severity issue was found and fixed with the update. 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.

The developer team has responded to the initial version of this report and updated the contracts. Most of the issues have been fixed including the high severity one.

The contracts are deployed to the mainnet of Binance Smart Chain:

0x4F3266a56589357B4f8082918b14B923693e57f0 LicoToken,

0x25eF2439ee92552B6f58463394E66D289F9e7b7C MasterChefV3,

0x6c3B1Ff7481fE0B106D046CfE60c5751caA30c7d LicoVault,

0xd05176C5A049C0ACF946D8992016dE4c9A9761a1 RewardTimer.

The Liquid Collectibles team contacted us after the deployment. The MAX_EMISSION_RATE constant in the MasterChefV3 contract that is used in the updateEmissionRate() function was set to 50 wei instead of 50e18 of LICO token. Thus, owners of MasterChefV3 contract should never call this function unless they want to effectively stop minting the rewards. The emission rate could only be decreased from its initial value via closed pools with positive allocation. Users can check the licoPerSecond value to be equal to 5e18.

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

8. References

1. Governance docs by OpenZeppelin
2. Dracula Protocol Medium
3. Address library by OpenZeppelin

HashEx

Blockchain Security