HashEx

Iceman Finance

smart contracts audit report

Prepared for:

iceman.finance

Authors: HashEx audit team

August 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 Iceman Finance team to perform an audit of their smart contracts. The audit was conducted between July 31 and August 05, 2021.

The code located in the github repository @icemanfinance/contracts was audited after the commit 169d489. It must be noted that only flats/MasterChef.sol and flats/Strategy_PCS.sol files were the subject of the audit. No documentation was provided. The repository contains only the contracts code without any tests or deploy scripts. The same contracts are deployed to the Binance Smart chain (BSC):
0xe919Ab6b749fdB3586829f29770985834b3A3541 MasterChef,
0x28E252BAA6C44a581B840061C1FE728b24933831 Strategy_PCS.

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: The Iceman 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 e0b2805 commit.

Contracts overview

`MasterChef`

Staking contract similar to MasterChef by SushiSwap [1] (which's audit is available [2,3]) with minor modifications. Transfers deposits further into Strategy contracts.

`Strategy_PCS`

Strategy contract for using the same tokens with 2 different Chefs simultaneously.

`SafeMath, ERC20, SafeERC20, Address, Ownable, ReentrancyGuard, Pausable`

Identical to OpenZeppelin's version 3.4.

Found issues

ID	Title	Severity	Response
01	MasterChef: emission rate not limited	High	Fixed
02	MasterChef: owner allowed to drain pools	High	Fixed
03	MasterChef: allowed duplicated pools	Medium	Responded
04	MasterChef: tokens with transfer fees are not supported	Medium	Responded
05	MasterChef: exceeding gas limit functions	Medium	Responded
06	MasterChef: referrals percent	Medium	Responded
07	MasterChef: new pool lasRewardBlock	Medium	Responded
08	MasterChef: massUpdate is optional	Medium	Responded
09	Strategy_PCS: router deadline	Medium	Acknowledged
10	Strategy_PCS: buybackRouter may not burn tokens	Medium	Acknowledged
11	MasterChef: startTime can be set in past	Medium	Acknowledged
12	MasterChef: maxSupply will be minted in a year	Low	Acknowledged
13	MasterChef: inconsistent comments	Low	Acknowledged
14	MasterChef: NATIVE address	Low	Acknowledged
15	MasterChef: declaring as constants	Low	Acknowledged
16	Multiple: check of uint <= 0	Low	Acknowledged
17	Multiple: lack of events	Low	Acknowledged
18	Multiple: declaring vars with 0 assignment	Low	Acknowledged
19	Strategy_PCS: optimizations in constructor	Low	Acknowledged
20	Strategy_PCS: excessive computations	Low	Acknowledged
21	Strategy_PCS: last user will withdraw less	Low	Acknowledged

22	Strategy_PCS: withdraw without earn	Low	Acknowledged
23	Strategy_PCS: input data not checked	Low	Acknowledged
24	General recommendations	Low	Acknowledged

#01 MasterChef: emission rate not limited — High

`updateEmissionRate()` function L1453 is used for updating the `NATIVEPerBlock` parameter. Although it's the onlyOwner function, we recommend adding safety guards — capping the new value. If the owner's account gets compromised or the owner acts maliciously, the attacker can set an arbitrary big value for the `NATIVEPerBlock` variable. In such a case the number of tokens till cap will be minted soon and the token price will drop. Moreover, it may become irreversible if an attacker adds new pools to the block gas limit of the `massUpdatePools()` function.

Recommendation: limit the `NATIVEPerBlock` parameter in `updateEmissionRate()` function.

Update: the issue was fixed.

#02 MasterChef: owner allowed to drain pools — High

inCaseTokensGetStuck() function L1435 allows the owner to transfer any tokens except the NATIVE from the contract's balance to the owner's wallet.

Recommendation: deny recovering of pool's `want` tokens.

Update: the issue was fixed.

#03 MasterChef: allowed duplicated pools — Medium

`add()` function L1145 allows adding a new pool with `want` token of the existing pool. It would mess up the rewards and wrong pools can't be removed or edited.

Iceman team response: if we would like to add another token or LP as a pool or farm from a different provider we wouldn't be able to do so. For example let's say we want to add BNB-BUSD LP from PancakeSwap and BNB-BUSD from ApeSwap, if we change this, we wouldn't be able to do so, that's why we have this setup.

Commentary on the update: owners should add the same tokens pools with extra caution and inform their users beforehand.

#04 MasterChef: tokens with transfer fees are not supported — Medium

MasterChef contract doesn't support pools of tokens with fees on transfers as transfers don't check the resulting balance.

Iceman team response: we don't have a deflationary token.

#05 MasterChef: exceeding gas limit functions — Medium

for() loop over the pools array in `massUpdatePools()` L1253 and `harvestAll()` L1294 functions may lead to block gas limit exceedance. Owners should monitor average gas costs of these functions and restrict adding the new pools in boundary conditions.

Iceman team response: the fee could be elevated when the user does a harvest all function, this is to avoid doing a manual harvest to all the farms we have. It benefits the end-user and makes the administrative tasks more efficient.

#06 MasterChef: referrals percent — Medium

`percReferrals` variable is set to 1 by default with the following comment: 'Referrals reward per block: 1%' L1091. But the denominator for this variable is 1e4, not the 1e2 (see L1302, L1332, L1378).

Iceman team response: we utilize a factor for every integer we have in the smart contract, this factor multiplies integers by 100, this is done to avoid extra decimals that could cause issues when compiling the contract. The number matches the referral percentage to 1%, however it may appear as 0.01% if the factor hasn't been accounted. 0.01%*100 = 1%

#07 MasterChef: new pool lastRewardBlock — Medium

`add()` function L1145 doesn't check the input value for `lastrewardBlock` of the new pool. It can be set prior to the `startBlock` of the contract which means the possible premine of the reward token.

Iceman team response: our launch is based on a date and not on a block.

Commentary on the update: setting `lastRewardBlock` of the new pool prior the `startTime` allows premine. We can't ensure if this is the desired behavior.

#08 MasterChef: massUpdate is optional — Medium

`add()` and `set()` functions have optional `_withUpdate` flag which calls `massUpdatePools()` if set true and may cause unfair rewards in case of rarely updated pools [4].

Iceman team response: this is to update all farms or multiple farms at once if needed, since we have several farms across all chains.

Commentary on the update: optional status of the `massUpdatePools()` allows the owner to abuse their power and may lead to unfair rewards distribution.

#09 Strategy_PCS: router deadline — Medium

`routerDeadlineDuration` variable is used to be added to `block.timestamp` in calls to the swap router. It means that the real swap translation could be mined at any time because the require during the swap checks `block.timestamp <= deadline` or `block.timestamp <= block.timestamp + routerDeadlineDuration` in our case.

#10 Strategy_PCS: buybackRouter may not burn tokens — Medium

`buybackRouterAddress` may be set to the custom router that translates the calls to the real router but doesn't burn the buybacked tokens. Users should pay attention to the values of both router addresses.

#11 MasterChef: startTime can be set in past — Medium

`inCaseOfRequiringChangeOfInitialDate()` function L1443 allows the owner to update the startTime variable only once but without filtering the new value.

Recommendation: require the current time to be less than the old value and the new value greater than the old one.

Update: the issue was fixed.

#12 MasterChef: maxSupply will be minted in a year — Low

`NATIVEMaxSupply` of 2e24 will be minted in slightly less than a year with the default emission rate. We can't ensure that this is the desired behavior.

#13 MasterChef: inconsistent comments — Low

Comments L1103, 1107, 1111 contain formulae for default values of emission rate which are slightly inaccurate with the errorabout 1e-14.

#14 MasterChef: NATIVE address Low

`NATIVE` token address could be declared as `NativeToken NATIVE`.

#15 MasterChef: declaring as constants Low

Multiple variables should be declared constants/immutables for gas savings:
`NATIVE`, `ownerNATIVEReward`, `percReferrals`, `NATIVEMaxSupply`, `NATIVEPerBlockBSC`, `NATIVEPerBlockFTM`, `NATIVEPerBlockMatic`. Constants should be named UPPERCASE [5].

#16 Multiple: check of uint <= 0 Low

Uint256 variable is checked whether it's less than 0 in L1270 of MasterChef and L1423 of Strategy_PCS.

#17 Multiple: lack of events Low

MasterChef contract emits events only with deposits and withdrawals, but not for updating state variables and adding/changing pools. Startegy_PCS has no event at all.

#18 Multiple: declaring vars with 0 assignment Low

Declaring variables with simultaneous 0 assignment L1100 of MasterChef and L1084-1086, 1125 of Strategy_PCS spends more gas than default declaring with the same result.

#19 Strategy_PCS: optimizations in constructor Low

Multiple unnecessary reads of state variables in constructor L1166-1197, function local parameters should be used to save gas on reads.

#20 Strategy_PCS: excessive computations Low

No need to check if `FeeFactor < FeeFactorMax` L1223, 1296 as it's always less or equal and the formula for `wantAmtWithFee` works with all possible values. Reading 2 state variables for ~1500 of gas to save ~10 (with another 2 reads) is useless.

#21 Strategy_PCS: last user will withdraw less Low

Last user may withdraw less amount as they would have less shares than totalShares at the moment due to possible dust of shares L1300.

#22 Strategy_PCS: withdraw without earn Low

`withdraw()` function L1274 with `autoComp = true` doesn't call `earn()` before the actual withdrawal. Thus, users may receive less than they are supposed to.

#23 Strategy_PCS: input data not checked Low

Constructor lacks the input data checks, making it harder to deploy the contracts.

#24 General recommendations Low

We strongly recommend putting the code through thorough testing as much as possible before release.

Conclusion

2 high severity issues were found. The audited code is located in the GitHub repository without any tests. We strongly recommend adding tests with coverage of at least 90% before the deployment to the mainnet. The contracts are highly dependent on the owner's account. Users of the project have to trust the owner and that the owner's account is properly secured.

This audit includes recommendations on the code improving and preventing potential attacks.

Update: Iceman Finance team has responded to this report. Most of the issues were fixed including all the high ones. Updated contracts are located in the same repository after the e0b2805 commit.

The audited contracts are deployed to the Binance Smart chain:
`0xe919Ab6b749fdB3586829f29770985834b3A3541` MasterChef,
`0x28E252BAA6C44a581B840061C1FE728b24933831` Strategy_PCS.

References

1. SushuSwap's MasterChef contract
2. SushiSwap audit by PeckShield
3. SushiSwap audit by Quantstamp
4. Dracula Protocol Medium
5. Solidity Docs: Naming Conventions

Appendix A. Issues' severity classification

We consider an issue to be 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