#HashEx
BLOCKCHAIN SECURITY

BNB Staking Club

smart contracts
final audit report

October 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).

2. Overview

HashEx was commissioned by the BNB Staking Club team to perform an audit of their smart contract. The audit was conducted between 06/09/2022 and 07/09/2022.

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 0x8Dcb14cFA5299BBe81eC873d71299b5A93991531 in the testnet of Binance Smart Chain (BSC).

Update: the BNB Staking Club team has responded to this report. The updated code is located at 0x016BB72F4b8174c4cA89E29c8D20C2f02c65F65A in the BSC mainnet.

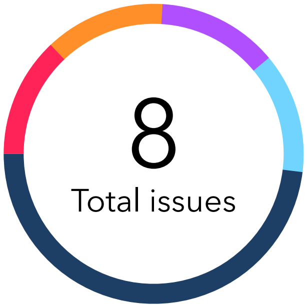
2.1 Summary

Project name	BNB Staking Club
URL	https://bnbstakingclub.com/
Platform	Binance Smart Chain
Language	Solidity

2.2 Contracts

Name	Address
BinanceStakingClub / BnbStakingClub	`0x016BB72F4b8174c4cA89E29c8D20C2f02c65F65A`

3. Found issues



● Critical	1 (13%)
● High	1 (13%)
● Medium	1 (13%)
● Low	1 (13%)
● Info	4 (48%)

Cf4. BinanceStakingClub / BnbStakingClub

ID	Severity	Title	Status
Cf4I73	● Critical	Threat of reentrancy	✓ Resolved
Cf4I70	● High	Exaggerated owner's rights	✓ Acknowledged
Cf4I74	● Medium	Questionable calculations	✓ Partially fixed
Cf4I6e	● Low	Gas optimizations	✓ Resolved
Cf4I75	● Info	Typos	✓ Resolved
Cf4I72	● Info	Wrong constants	✓ Resolved
Cf4I71	● Info	Threat of the owner's front-run	✓ Resolved
Cf4I6d	● Info	Lack of events	✓ Partially fixed

4. Contracts

Cf4. BinanceStakingClub / BnbStakingClub

Overview

Simple staking contract that accepts the user's native currency and emits the corresponding event. All the funds are withdrawable by the owner regardless the staking parameters. User may request an early withdrawal, a claim profit, and unstake after the staking period has passed, but there's no guarantee that the contract has enough funds to fulfill the request. Source of the rewards is unclear and out of the scope of this audit.

Issues

Cf4I73 Threat of reentrancy

● Critical ✓ Resolved

Functions `earlyLockupWithdrawal()`, `regularWithdrawal()`, and `profitWithdrawal()` are vulnerable to reentrancy attacks. A user can withdraw all funds from the contract.

```
function regularWithdrawal(uint numberOfStaking) external {
    ...
    require(staking.isFinished == false, 'This staking is finished');

    address payable to = payable(_msgSender());
    (bool success, ) = to.call{value:withdrawAmount}("");
    require(success, "Transfer failed.");

    staking.isFinished = true;
    ...
}
```

This vulnerability is exploitable instantly since the `stake()` function can be called with `numberOfMonths = 0`.

Recommendation

Add the modifier `nonReentrant()` from OpenZeppelin's ReentrancyGuard to these functions.

Cf4I70 Exaggerated owner's rights

● High ✓ Acknowledged

The contract's owner has the access to all the collected funds regardless of the staking periods.

```
function withdrawAll() payable external onlyOwner {
    address payable to = payable(_msgSender());
    address thisContract = address(this);
    (bool success, ) = to.call{value:thisContract.balance}("");
    require(success, "Transfer failed.");
}

function transfer(address payable to, uint256 amount) external nonReentrant onlyOwner {
    require(address(this).balance >= amount,  'Transfer amount exceeds balance');

    (bool success, ) = to.call{value:amount}("");
    require(success, "Transfer failed.");
}
```

Moreover, the owner can set any amount of referral income because the owner sets this value manually. Only the owner can withdraw someone's referral income.

```
function withdrawalReferralIncome(address payable recipient, uint256 amount) external
nonReentrant onlyOwner {
    require(address(this).balance >= amount,  'Transfer amount exceeds balance');
    require(profile[recipient].availableReferralIncome >= amount,  'Transfer amount
exceeds his income!');

    (bool success, ) = recipient.call{value:amount}("");
    require(success, "Transfer failed.");

    profile[recipient].availableReferralIncome =
profile[recipient].availableReferralIncome - amount;
}
```

```
function setReferralIncomeOfProfile(address recipient, uint256 income) external
nonReentrant onlyOwner {
    profile[recipient].availableReferralIncome = income;
}
```

Recommendation

We strongly recommend restricting the owner's ability to manipulate staked funds. It is a good practice to use a Timelock contract as an owner of the BinanceStakingClub contract with a MultiSig wallet as an admin of the Timelock contract.

Team response

The reason for setting it up in that way is that we need to have access to funds as they will be distributed among other big protocols on BNB Smart Chain, like Pancakeswap, Beefy Finance, Venus, GrizzlyFi, *etc*., in order to generate yield for BSC users.

Cf4I74 Questionable calculations ● Medium Partially fixed

Profit calculation in the `profitAmountCalculation()` function raises some questions.

On the line 242 the result of the division would always be zero.

```
availableWithdrawalAmount = amount * (1 + coefficient / 1000) ** daysDiff - amount;
```

It isn't clear what is happening with calculations of `availableWithdrawalAmount` in general, no documentation was presented.

It seems that the early withdrawal penalty is 10%, but staking for any period greater that 12 increases profit by 3 times compared to 0-5 months of locking. There's no point to lock with `numberOfMonths < 12`, as bonus coefficient for lock period prevails the penalty.

It also seems that the `isAutoCompound` staking flag affects only the profit calculation. There's no point for user to stake with `isAutoCompound == false`, as the profit in both cases could be make only from external sources. Moreover, `lockupFeeCoefficient` (coefficient for the penalty for early withdrawal) is applied only in case `isAutoCompound == false`.

Recommendation

We strongly recommend adding functional and unit tests with at least 80% coverage and provide adequate documentation for the users.

Update

The calculation of user's profit leads to APY of approximately 787% in case of auto compound and 219% without it. These numbers seem questionable. In the tests the upcoming profit is covered by the staked funds of another users.

Cf4I6e Gas optimizations ● Low ✓ Resolved

Several optimizations could be made to reduce gas on both usage and deployment costs:

1. In the struct `Staking` fields `isFinished` and `profitWithdrawalRequested` can be swapped to make the packing of the structure in the storage more gas-efficient.
2. In functions `earlyLockupRequest()`, `earlyLockupWithdrawal()`, `regularWithdrawal()`, `profitWithdrawalRequest()`, and `profitWithdrawal()` the whole structure `Staking` is read for a sender, then some fields are changed, and the entire structure is written back into the storage. This process can be made more efficient when the number of fields of the structure that are read (in the whole transaction) is less than 6 (in optimized version 5 (see the first item in this list)) by making separate readings for each field. This will reduce the amount of SLOAD operations. Also, only fields that are changed can be written back into the storage. This will reduce the amount of STORE operations because the number of changed fields is less than 6.
3. In function `withdrawalReferralIncome()` global variable `profile[recipient].availableReferralIncome` is read multiple times.
4. The field `from` in the structure `Staking` is redundant.

5. In function `profitWithdrawal()` global variable `profile[_msgSender()].stakings[numberOfStaking].profitWithdrawalRequested` is read multiple times.

Update

1. `uint8` and `bool` fields of the `Staking` struct could be placed adjacent to minimize the storage needed.

2. The `profitAmountCalculation()` function still reads the whole `Staking` struct besides some parts of it are needed only in contradictory if/else conditions.

3rd and 4th items of the list were fixed.

Cf4I75 Typos ● Info ✓ Resolved

Typos reduce the code's readability.

1) L42L 'Mininmum' should be replaced with 'Minimum'

2) L255 'Refferal' should be replaced with 'Referral'

3) L397 'Withrawal' should be replaced with 'Withdrawal'

Update

1st and 2nd elements were fixed.

Third element was introduced with the update.

Cf4I72 Wrong constants ● Info ✓ Resolved

In the function `isEarlyWithdrawal()` the constant `30 minutes` on the line 83 should be replaced with `30 days`.

In the function `isEarlyWithdrawalRequestLocked()` the constant `3 minutes` on the line 89 should be replaced with `3 days`, presumably.

In the function `isProfitWithdrawalRequestLocked()` the constant `1 minutes` on the line 95 should be replaced with `1 days`, presumably.

In the function `profitAmountCalculation()` the constant `60` on the line 216 should be replaced `1 days`. with

Update

The mentioned constants were fixed with the update, but 1 new testnet constant was introduced: `0.02e18` on line 49 should be replaced with `0.2e18` for the mainnet launch.

Cf4I71 Threat of the owner's front-run ● Info ✓ Resolved

If a user urgently wants to withdraw his funds, he won't be able to withdraw them faster than 3 minutes (see the 'Wrong constants' issue). For example, if a user requested an early lockup request, the contract sends an appropriate event, and the owner sees it and makes a withdrawal of the contract's funds before the user performs an early lockup withdrawal. Because of that, the user won't be able to perform a withdrawal of his funds.

The same with profit withdrawal requests and profit withdrawals.

Recommendation

It is a good practice to use a Timelock contract as an owner of the BinanceStakingClub contract with MultiSig wallet as an admin of the Timelock contract.

Cf4l6d Lack of events ● Info ✓ Partially fixed

All governance `onlyOwner` functions should emit specific events, complicating the off-chain tracking of changes.

Update

`setReferralIncomeOfProfile()` doesn't emit any events.

Also, `backendAddress` and `gnosisSafeAddress` don't have default visibility, nor external viewers.

5. Conclusion

1 critical, 1 high, 1 medium, 1 low severity issues were found during the audit. 1 critical, 1 low issues were resolved in the update.

The reviewed contract is extremely dependent on the owner’s account. Users using the project have to trust the owner and that the owner's account is properly secured.

We strongly recommend adding functional and unit tests with at least 80% coverage and provide adequate documentation for the users. Test were provided with the updated contract.

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.
- **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

✉ contact@hashex.org

@hashex_manager

blog.hashex.org

linkedin

github

twitter

HashEx
BLOCKCHAIN SECURITY