



QuillAudits



Audit Report  
April, 2021



BasketCoin



# Contents

Introduction	01
Techniques and Methods	02
Issue Categories	04
Issues Found – Code Review/Manual Testing	06
Automated Testing	07
Closing Summary	08
Disclaimer	09

## Overview

### BsktLPPool

**BsktLPPool** is the staking pool for BasketCoin

- Pool Duration is 30 days, which can be updated by the Owner
- Initial Staking is 5000 tokens per address/user
- 2% token difference on the staked amount
- No Time lock: Users can exit and withdraw any time

**Contract:** BsktLPPool.sol

**Description Report:** BsktLPPool.md

### Scope of Audit

The scope of this audit was to analyse **BsktLPPool.sol** smart contract's codebase for quality, security, and correctness.

### Checked Vulnerabilities

We have scanned the BasketCoin smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Signature Malleability
- EIP72 Structures
- Replay Attacks
- Use of ecrecover() function
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply



- Integer overflow/underflow
- ERC20 transfer() does not return
- boolean
- ERC20 approve() race
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

## Techniques and Methods

Throughout the audit of BasketCoin smart contracts care was taken to ensure:

- The Overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per intended behaviour mentioned in whitepaper.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

### Structural Analysis


In this step we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

### Static Analysis

Static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step a series of automated tools are used to test security of smart contracts.

### Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerability or verify the vulnerabilities found during the static analysis. Contracts were



completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of automated analysis were manually verified.

## **Gas Consumption**

In this step we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and possibilities of optimization of code to reduce gas consumption.

## **Tools and Platforms used for Audit**

Mythril, Slither, SmartCheck, Solhint.





# Issue Categories

Every issue in this report has been assigned with a severity level. There are four levels of severity and each of them has been explained below.

## High severity issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract’s performance or functionality and we recommend these issues to be fixed before moving to a live environment.

## Medium level severity issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems and they should still be fixed.

## Low level severity issues

Low level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

## Informational

These are severity four issues which indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

## Number of issues per severity

	High	Medium	Low	Informational
Open	0	0	3	1
Closed	0	1	2	1



# Issues Found – Code Review / Manual Testing

## High severity issues

None.

## Medium severity issues

- [FIXED]Reentrancy Checks as mentioned in the Slither Report(provided in the automated tests section)

The best practices to avoid Reentrancy weaknesses are:

-Make sure all internal state changes are performed before the call is executed. This is known as the Checks-Effects-Interactions pattern

<https://docs.soliditylang.org/en/latest/security-considerations.html#use-the-checks-effects-interactions-pattern>

-Use a reentrancy lock (ie. OpenZeppelin's ReentrancyGuard.

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/security/ReentrancyGuard.sol>

Updates: Reentrancy Guard has been implemented

## Low level severity issues

- [#L9] Old solidity compiler used: Use the latest compiler versions to avoid bugs found in old compilers
- [#L9] Floating Pragma: Different Pragma Directives have been used
- [FIXED][#L350]Unused state variable **rewardAmount**
- [FIXED][#L356-359] Unused Events.
- [#L363, 380] Use of **block.timestamp** for comparisons. Avoid using block.timestamp as it can be manipulated by miners.

## Informational

- [FIXED][#L110] Use `_msgSender()`; function instead of using `msg.sender`; directly
- [#453-455] Event can be added to notify Duration change

## Remix IDE

### Compiling with solc < 0.5.5

**Warning:** The "extcodehash" instruction is not supported by the VM version "byzantium" you are currently compiling for. It will be interpreted as an invalid instruction on this VM.

## Gas Optimization

Public functions that are never called by the contract could be declared external to save gas.

```
owner() should be declared external:
- Ownable.owner() (BsktLPPool.sol#114-116)
renounceOwnership() should be declared external:
- Ownable.renounceOwnership() (BsktLPPool.sol#127-130)
transferOwnership(address) should be declared external:
- Ownable.transferOwnership(address) (BsktLPPool.sol#132-134)
notifyRewardRate(uint256) should be declared external:
- BsktLPPool.notifyRewardRate(uint256) (BsktLPPool.sol#442-450)
setDuration(uint256) should be declared external:
- BsktLPPool.setDuration(uint256) (BsktLPPool.sol#453-455)
```



# Automated Testing

## Slither

```
Reentrancy in BsktLPPool.exit() (BsktLPPool.sol#425-428):
  External calls:
  - withdraw(balanceOf(msg.sender)) (BsktLPPool.sol#426)
    - BSKTASREWARD.safeTransfer(msg.sender,amount) (BsktLPPool.sol#337)
    - (success, returndata) = address(token).call(data) (BsktLPPool.sol#292)
  - getReward() (BsktLPPool.sol#427)
    - (success, returndata) = address(token).call(data) (BsktLPPool.sol#292)
    - STAKEBSKT.safeTransfer(msg.sender, reward) (BsktLPPool.sol#436)
  State variables written after the call(s):
  - getReward() (BsktLPPool.sol#427)
    - lastUpdateTime = lastTimeRewardApplicable() (BsktLPPool.sol#371)
  - getReward() (BsktLPPool.sol#427)
    - rewardPerTokenStored = rewardPerToken() (BsktLPPool.sol#370)
  - getReward() (BsktLPPool.sol#427)
    - rewards[msg.sender] = 0 (BsktLPPool.sol#434)
    - rewards[account] = earned(account) (BsktLPPool.sol#373)
  - getReward() (BsktLPPool.sol#427)
    - userRewardPerTokenPaid[account] = rewardPerTokenStored (BsktLPPool.sol#374)
Reentrancy in BsktLPPool.stake(uint256) (BsktLPPool.sol#405-414):
  External calls:
  - super.stake(amount) (BsktLPPool.sol#411)
    - (success, returndata) = address(token).call(data) (BsktLPPool.sol#292)
    - BSKTASREWARD.safeTransferFrom(msg.sender, address(this), amount) (BsktLPPool.sol#331)
  State variables written after the call(s):
  - minimumBsktStakingEntry[_msgSender()] = true (BsktLPPool.sol#413)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
```

## Mythril

Mythril detected SWC-116 (block.timestamp dependence issue) as mentioned in low-severity issues above

## Smartcheck

Smartcheck didn't detect any high severity issues.

## Solhint

```
BsktLPPool/BsktLPPool.sol
310:2  error  Line length must be no more than 120 but current length is 132  max-line-length
345:2  error  Line length must be no more than 120 but current length is 133  max-line-length

× 2 problems (2 errors, 0 warnings)
```



## Closing Summary

Overall, smart contracts are very well written and adhere to guidelines. During the process of audit, No issues of high, and medium severity were recorded. However, some low severity issues were found and have been documented above.



## Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the **BsktLPPool Platform**. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the **BsktLPPool Team** put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.





# BasketCoin



## QuillAudits



Canada, India, Singapore and United Kingdom



[audits.quillhash.com](https://audits.quillhash.com)



[hello@quillhash.com](mailto:hello@quillhash.com)