QuillAudits

Audit Report
May, 2021

BasketCoin

Contents

Overview

BsktLPPool

BsktEthLPPool is the staking pool for BasketCoin

- Pool Duration is 100 days and fixed.
- No Minimum Amount Limitation for initial staking entry.
- No token difference on the staked amount
- Reward Rate: 1Million over 100 days, i.e., 10K BSKT everyday
- No Time lock: Users can exit and withdraw any time

Contract: BsktEthLPPool.sol

Description Report: BsktEthLPPool.md

Scope of Audit

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

Checked Vulnerabilities

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

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- 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

- Integer overflow/underflow
- ERC20 transfer() does not return boolean
- ERC20 approve() race
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of 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 the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- 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.
SmartCheck.

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, Surya, 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	2	0
Closed	0	1	1	0

Issues Found – Code Review / Manual Testing

High severity issues

None.

Medium severity issues

1. **[FIXED]** [#L459-462] function **exit()** should not be nonReentrant

Low Severity Issues

1. [#L7] Old solidity compiler used: Use the latest compiler versions to avoid bugs found in old compilers

2. [#L400, 417] Use of **block.timestamp** for comparisons. Avoid using block.timestamp as it can be manipulated by miners.

2. **[FIXED] Compiling with solc 0.5.0**
   [#L224]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.

   Reason: Changed the solc version to 0.5.17

Informational

None.

- **Gas Optimization-** public functions that are never called by the contract could be declared external to save gas.

Automated Testing

Slither

```
Reentrancy in BsktEthLPPool.exit() (BsktEthLPPool.sol#459-462):
        External calls:
        - withdraw(balanceOf(_msgSender())) (BsktEthLPPool.sol#460)
                - BSKTASREWARD.safeTransfer(_msgSender(),amount) (BsktEthLPPool.sol#373)
                - (success,returndata) = address(token).call(data) (BsktEthLPPool.sol#333)
        - getReward() (BsktEthLPPool.sol#461)
                - STAKEBSKT.safeTransfer(_msgSender(),reward) (BsktEthLPPool.sol#471)
                - (success,returndata) = address(token).call(data) (BsktEthLPPool.sol#333)
        State variables written after the call(s):
        - getReward() (BsktEthLPPool.sol#461)
                - _status = _ENTERED (BsktEthLPPool.sol#42)
                - _status = _NOT_ENTERED (BsktEthLPPool.sol#48)
        - getReward() (BsktEthLPPool.sol#461)
                - lastUpdateTime = lastTimeRewardApplicable() (BsktEthLPPool.sol#408)
        - getReward() (BsktEthLPPool.sol#461)
                - rewardPerTokenStored = rewardPerToken() (BsktEthLPPool.sol#407)
        - getReward() (BsktEthLPPool.sol#461)
                - rewards[_msgSender()] = 0 (BsktEthLPPool.sol#469)
                - rewards[account] = earned(account) (BsktEthLPPool.sol#410)
        - getReward() (BsktEthLPPool.sol#461)
                - userRewardPerTokenPaid[account] = rewardPerTokenStored (BsktEthLPPool.sol#411)
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

```
 351:2  error  Line length must be no more than 120 but current length is 137  max-line-length
 382:2  error  Line length must be no more than 120 but current length is 134  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 severity were recorded. Some low, and medium severity issues were found and have been documented above. However, most of them have been resolved and tested.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the **BsktEthLPPool 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 **BsktEthLPPool** 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

hello@quillhash.com