BSCS Smart Contract Initial Audit Report

Scope of Audit

The scope of this audit was to analyze and document BSCS smart contract codebase for quality, security, and correctness.

Check Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

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.

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 the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities 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 the 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

Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

Issue Categories

Every issue in this report has been assigned to 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 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 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 four severity 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

TYPE	HIGH	MEDIUM	LOW	INFORMATIONAL
OPEN	0	2	2	1
CLOSED	0	0	0	0

Introduction

During the period of **April 5th, 2021 to April 8th, 2021** - Quillhash Team performed a security audit for **BSCS** smart contracts.

The code for the audit was taken from following the official Github link**:**

https://github.com/BSCStationSwap/bscsswap-exchange/blob/main/contract/Caketoken.sol

Commit Hash - *191baab86a4a64d48b4c320f4e76593a88320eaa*

A. Contract Name - CakeToken

Issues Found – Code Review / Manual Testing

High Severity Issues

Not Found

Medium Severity Issues

A.1 Require Statement should be preferred instead of IF Statement

Line no - 1045

Description:

The **_moveDelegates** function ensures that that the function is executed only if:

- The **srcRep** address and the **dstRep** address should be different, and
- The amount passed should be greater than **Zero.**

According to the current function design, if any of these conditions is not fulfilled, the function will not be executed.

Therefore, keeping in mind that the function strictly checks the arguments passed and stops the function execution if the above-mentioned conditions are not met, it would be more effective to use a **require statement** instead of **IF-Else Statement.**
Its not only considered a better practice to use **require statements** for input validation but it also improves the code readability and helps in gas optimization.

Recommendation:

Use **require** instead of **IFStatement.**

A.2 Strict Equality is being used in the IF statement
Line no: 122-124

Description:
During the automated testing of the CakeToken Contract, it was found that the **_writeCheckpoint** function includes a STRICT EQUALITY check in the if statement.

```
1073
1074        if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber) {
1075            checkpoints[delegatee][nCheckpoints - 1].votes = newVotes;
```

Is this Intended?

Recommendation:
It is not considered a better practice in Solidity to implement a *Strict Equality* check in the **If** or **require** statements.
If the above-mentioned logic is not intended, then the **if statement** should be modified.

Low Severity Issues

A.2 External visibility should be preferred
Line no: 860

Description:
Those functions that are never called throughout the contract should be marked as ***external*** visibility instead of ***public*** visibility.
This will effectively result in Gas Optimization as well.
Therefore, the following function must be marked as **external** within the contract:

- ***mint***

Recommendation:
The above-mentioned functions should be assigned external visibility.

A.14 NatSpec Annotations must be included

Description:
Smart contract does not include the NatSpec annotations adequately.

Recommendation:
Cover by NatSpec all Contract methods.

Informational

B.5 Coding Style Issues

Description:
Coding style issues influence code readability and in some cases may lead to bugs in future. Smart Contracts have a naming convention, indentation and code layout issues.

The order of functions as well as the rest of the code layout does not follow the solidity style guide.

Layout contract elements in the following order:

a. Pragma statements

b. Import statements

c. Interfaces

d. Libraries

e. Contracts

Inside each contract, library or interface, use the following order:

a. Type declarations

b. State variables

c. Events

d. Functions

Recommendations:
Please read the following documentation links to understand the correct order: - https://docs.soliditylang.org/en/v0.6.12/style-guide.html#order-of-functions

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines.

During the process of audit, No issues of high severity were found. Moreover, No instances of Re-entrancy or Back-Door Entry were found in the contract.

However, some medium as well as low severity were found which might affect the intended behaviour of the contracts. The issues have been explained in detail above and it is recommended to go through the details and update the contract accordingly.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the **BSCS 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 **BSCS** Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.