Magic Cube Smart Contract Initial Audit Report

Scope of Audit 2

Check Vulnerabilities 2

Techniques and Methods 3

Issue Categories 4

Number of security issues per severity. 5

Introduction 5

A. Contract – TokenInfo 6

Issues Found – Code Review / Manual Testing 6

High Severity Issues 6

Medium Severity Issues 6

Low Severity Issues 6

A.1 Approve Race: 6

A.2 Renounce Ownership: 7

A.3 Floating Pragma: 7

Informational 8

A.4 Function that can be declared external: 8

Functional Testing 9

Automated Tests 10

Slither: 10

Results: 11

Scope of Audit

The scope of this audit was to analyze and document the Magic Cube smart contracts 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 behavior.
- Token distribution and calculations are as per the intended behavior 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
A 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 behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Slither.

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 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 Issues

These are four severity issues that 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 security issues per severity.

TYPE	HIGH	MEDIUM	LOW	INFORMATIONAL
Open	0	0	3	1
Acknowledged	0	0	0	0
Closed	0	0	0	0

Introduction

During the period of **February 8, 2022, to February 10, 2022 – QuillAudits** Team performed a security audit for **Magic Cube** smart contract.

The code for the audit was taken from the contract address of **Magic Cube:**

https://bscscan.com/address/0xB31FD05CAF496CEf34C945667E90dd89C20E0D09#code

V	Date	Contract Address	Note
1	February	0xb31fd05caf496cef34c945667e90dd89c20e0d09	Version 1

A. Contract – TokenInfo

Issues Found – Code Review / Manual Testing

High Severity Issues

No issues were found.

Medium Severity Issues

No issues were found.

Low Severity Issues

A.1 Approve Race:

```
Line 464:
    function approve(address spender, uint256 amount) public virtual override returns (bool){
        _approve(_msgSender(), spender, amount);
        return true;
    }
```

Description:

The standard ERC20 implementation contains a widely-known racing condition in its approve function, wherein a spender is able to witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender will be able to get both approval amounts of both transactions.

Remediation:

It's recommended to use the increaseAllowance and decreaseAllowance functions to override the approved amount instead of using the approve function.

Status: Open

A.2 Renounce Ownership:

```
Line 888:
      contract TokenInfo is ERC20, ERC20Burnable, ERC20Pausable, Ownable {
```

Description:

Typically, the contract's owner is the account that deploys the contract. As a result, the owner is able to perform certain privileged activities on his behalf. The renounceOwnership function is used in smart contracts to renounce ownership. Otherwise, if the contract's ownership has not been transferred previously, it will never have an Owner, which is risky.

Remediation:

It is recommended that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method for two or more users should be confirmed. Alternatively, the RenounceOwnership functionality can be disabled by overriding it.

Status: Open

A.3 Floating Pragma:

```
Line 884:
   pragma solidity >=0.6.0 <0.8.0;
```

Description:

The contract makes use of the floating-point pragma 0.6.0 to 0.8.0 Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensuring that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation:

Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status: Open

Informational

A.4 Function that can be declared external:

Description:
The following functions can be declared external instead of public in order to save gas.
The functions:

name()
symbol()
decimals()
totalSupply()
balanceOf(address)
transfer(address,uint256)
approve(address,uint256)
transferFrom(address,address,uint256)
increaseAllowance(address,uint256)
decreaseAllowance(address,uint256)
burn(uint256)
burnFrom(address,uint256)
renounceOwnership()
transferOwnership(address)
setBlacklist(address,bool)
pause()
unpause()

Remediation:
Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status: Open

Functional Testing

constructor	PASS	PASS	PASS	
_beforeTokenTransfer	PASS	PASS	PASS	
setBlacklist	PASS	PASS	PASS	
pause	PASS	PASS	PASS	
unpause	PASS	PASS	PASS	
ERC20				
name	PASS	PASS	PASS	
symbol	PASS	PASS	PASS	
decimals	PASS	PASS	PASS	
_ totalSupply				
balanceOf	PASS	PASS	PASS	
transfer	PASS	PASS	PASS	
allowance	PASS	PASS	PASS	
approve	PASS	PASS	PASS	
transferFrom	PASS	PASS	PASS	
increaseAllowance	PASS	PASS	PASS	
decreaseAllowance	PASS	PASS	PASS	
_transfer	PASS	PASS	PASS	
_mint	PASS	PASS	PASS	
_burn	PASS	PASS	PASS	
_approve	PASS	PASS	PASS	
_setupDecimals	PASS	PASS	PASS	
_beforeTokenTransfer	PASS	PASS	PASS	

Automated Tests

Slither:

```
Context._msgData() (token.sol#106-109) is never used and should be removed
SafeMath.div(uint256,uint256) (token.sol#248-251) is never used and should be removed
SafeMath.div(uint256,uint256,string) (token.sol#303-306) is never used and should be removed
SafeMath.mod(uint256,uint256) (token.sol#265-268) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (token.sol#323-326) is never used and should be removed
SafeMath.mul(uint256,uint256) (token.sol#229-234) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (token.sol#137-141) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (token.sol#173-176) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (token.sol#183-186) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (token.sol#158-166) is never used and should be removed
SafeMath.trySub(uint256,uint256) (token.sol#148-151) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version0.6.0 (token.sol#884) allows old versions
solc-0.6.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Parameter TokenInfo.setBlacklist(address,bool)._toBlacklist (token.sol#908) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (token.sol#107)" inContext (token.sol#101-110)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

name() should be declared external:
        - ERC20.name() (token.sol#394-396)
symbol() should be declared external:
        - ERC20.symbol() (token.sol#402-404)
decimals() should be declared external:
        - ERC20.decimals() (token.sol#419-421)
totalSupply() should be declared external:
        - ERC20.totalSupply() (token.sol#426-428)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (token.sol#433-435)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (token.sol#445-448)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (token.sol#464-467)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (token.sol#482-486)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (token.sol#500-503)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (token.sol#519-522)
burn(uint256) should be declared external:
        - ERC20Burnable.burn(uint256) (token.sol#660-662)
burnFrom(address,uint256) should be declared external:
        - ERC20Burnable.burnFrom(address,uint256) (token.sol#675-680)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (token.sol#862-865)
transferOwnership(address) should be declared external:
```

```
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (token.sol#862-865)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (token.sol#871-875)
setBlacklist(address,bool) should be declared external:
        - TokenInfo.setBlacklist(address,bool) (token.sol#908-910)
pause() should be declared external:
        - TokenInfo.pause() (token.sol#913-915)
unpause() should be declared external:
        - TokenInfo.unpause() (token.sol#917-919)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
. analyzed (9 contracts with 75 detectors), 32 result(s) found
```

Results:

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines.
Many issues were discovered during the initial audit; it is recommended to fix them.

Disclaimer

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