Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	BNB Token
Website:	bnbchain.org
Platform:	Ethereum
Language:	Solidity
Date:	May 8th, 2024

Table of contents

Introduction 4

Project Background 4

Audit Scope 5

Claimed Smart Contract Features 6

Audit Summary 7

Technical Quick Stats 8

Business Risk Analysis 9

Code Quality 10

Documentation 10

Use of Dependencies 10

AS-IS overview 11

Severity Definitions 12

Audit Findings 13

Conclusion 18

Our Methodology 19

Disclaimers 21

Appendix

- Code Flow Diagram 22
- Slither Results Log 23
- Solidity static analysis 24
- Solhint Linter 25

`

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the smart contracts of BNB Token from BNBChain.org were audited. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on May 8th, 2024.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

- The BNB Token is native coin of the BNB Chain ecosystem. It is a utility token needed to use the BNB Chain platform.
- This Solidity contract is a token contract for a token named BNB. Here's a summary of its functionality:
  - **SafeMath Library:** This contract includes a library called SafeMath which provides functions for safe arithmetic operations to prevent overflows and underflows.
  - **BNB Token Contract:** The BNB contract implements the functionality of the BNB token.
  - **Token Properties:** It defines properties such as name, symbol, decimals, and totalSupply for the token.
  - **Balances:** It maintains a mapping of token balances for each address (balanceOf), and another mapping to track frozen token balances (freezeOf).
  - **Allowance:** It allows another address to spend tokens on behalf of the token owner by using the approve function.
  - **Token Transfer:** Functions transfer and transferFrom are implemented for transferring tokens between addresses. These functions include checks to ensure that the sender has sufficient balance and that there are no overflows.
  - **Token Burning:** The burn function allows token holders to burn (destroy) their own tokens, reducing the total supply.

    - **Token Freezing:** It provides functions freeze and unfreeze to freeze and unfreeze tokens for a particular address. Frozen tokens cannot be transferred until they are unfrozen.
    - **Withdraw Ether:** The withdrawEther function allows the contract owner to withdraw any Ether balance held by the contract.
    - **Fallback Function:** The contract includes a payable fallback function to accept Ether transfers.
- Overall, this contract provides the basic functionalities expected from an ERC-20 compatible token, along with additional features like token freezing and burning.

Audit scope

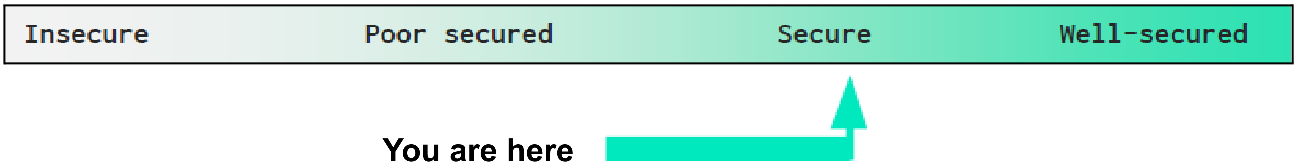
Name	**Code Review and Security Analysis Report for BNB Token Smart Contract**
Platform	**Ethereum**
Language	**Solidity**
File	BNB.sol
Smart Contract Code	0xB8c77482e45F1F44dE1745F52C74426C631bDD52
Audit Date	May 8th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: - Name: BNB - Symbol: BNB - Decimals: 18	**YES, This is valid.**
Ownership Control: - **withdrawEther**: Transfer BNB balance to owner.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. This token contract has ownership control. **It is ideal to renounce ownership once its purpose is over to make it fully decentralized.**



We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 0 low and 4 very low level issues.

Investors Advice: Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	"Out of Gas" Issue	Moderated
	High consumption 'for/while' loop	Passed
	High consumption 'storage' storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	"Short Address" Attack	Passed
	"Double Spend" Attack	Passed

Overall Audit Result: PASSED

Business Risk Analysis

Category	Result
🟢 Buy Tax	0%
🟢 Sell Tax	0%
🟢 Cannot Buy	No
🟢 Cannot Sell	No
🟢 Max Tax	0%
🟢 Modify Tax	Not Detected
🟢 Fee Check	No
🟢 Is Honeypot	Not Detected
🟢 Trading Cooldown	Not Detected
🟢 Can Pause Trade?	No
🟢 Pause Transfer?	Not Detected
🟢 Max Tax?	No
🟢 Is it Anti-whale?	Not Detected
🟢 Is Anti-bot?	Not Detected
🟢 Is it a Blacklist?	Not Detected
🟢 Blacklist Check	No
🟢 Can Mint?	No
🟢 Is it Proxy?	Not Detected
🟢 Can Take Ownership?	Not Detected
🟢 Hidden Owner?	Not Detected
🟢 Self Destruction?	Not Detected
🟢 Auditor Confidence	High

Overall Audit Result: PASSED

Code Quality

This audit scope has 1 smart contract. Smart contract contains Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in BNB Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the BNB Token.

The EtherAuthority team has no scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a BNB Token smart contract code in the form of Etherscan web link.

As mentioned above, code parts are well commented on. And the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well known industry standard open source projects.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	BNB	write	Passed	No Issue
3	transfer	write	Passed	No Issue
4	approve	write	Gas optimization	Refer Audit Findings
5	transferFrom	write	Gas optimization	Refer Audit Findings
6	burn	write	Gas optimization	Refer Audit Findings
7	freeze	write	Gas optimization, No Rate Limiting or Lock Period for Freeze/Unfreeze	Refer Audit Findings
8	unfreeze	write	Gas optimization, No Rate Limiting or Lock Period for Freeze/Unfreeze	Refer Audit Findings
9	withdrawEther	write	Centralized Risk, Gas optimization	Refer Audit Findings

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

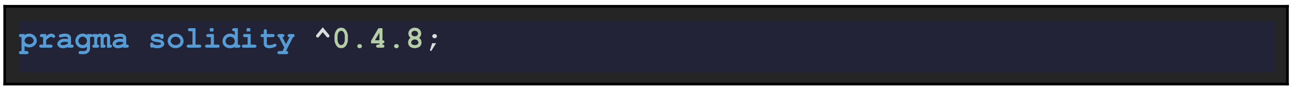
No Medium severity vulnerabilities were found.

Low

No Low severity vulnerabilities were found.

Very Low / Informational / Best practices:

(1) Use latest solidity version:

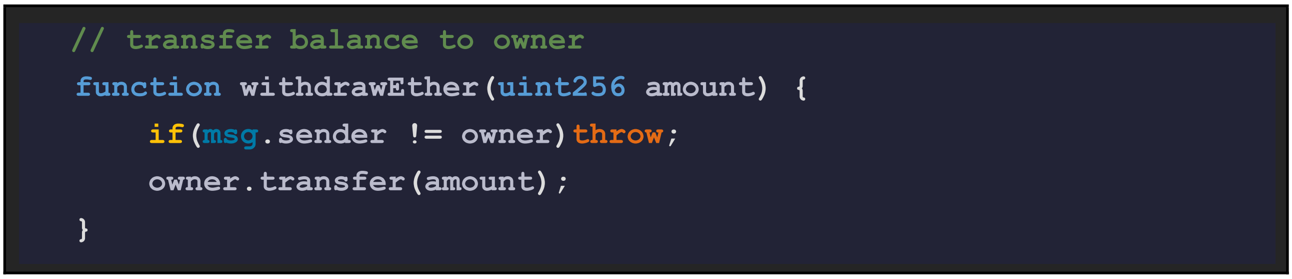


```
pragma solidity ^0.4.8;
```

Use the latest solidity version while contract deployment to prevent any compiler version level bugs.

Resolution: Please use the latest solidity versions.

(2) Centralization Risk:



```
    // transfer balance to owner
    function withdrawEther(uint256 amount) {
        if(msg.sender != owner)throw;
        owner.transfer(amount);
    }
```

The owner can take the fund out of the contract. This does not create issues in most cases. However, it is ideal to renounce the ownership once its purpose is over to make the contract fully decentralized.

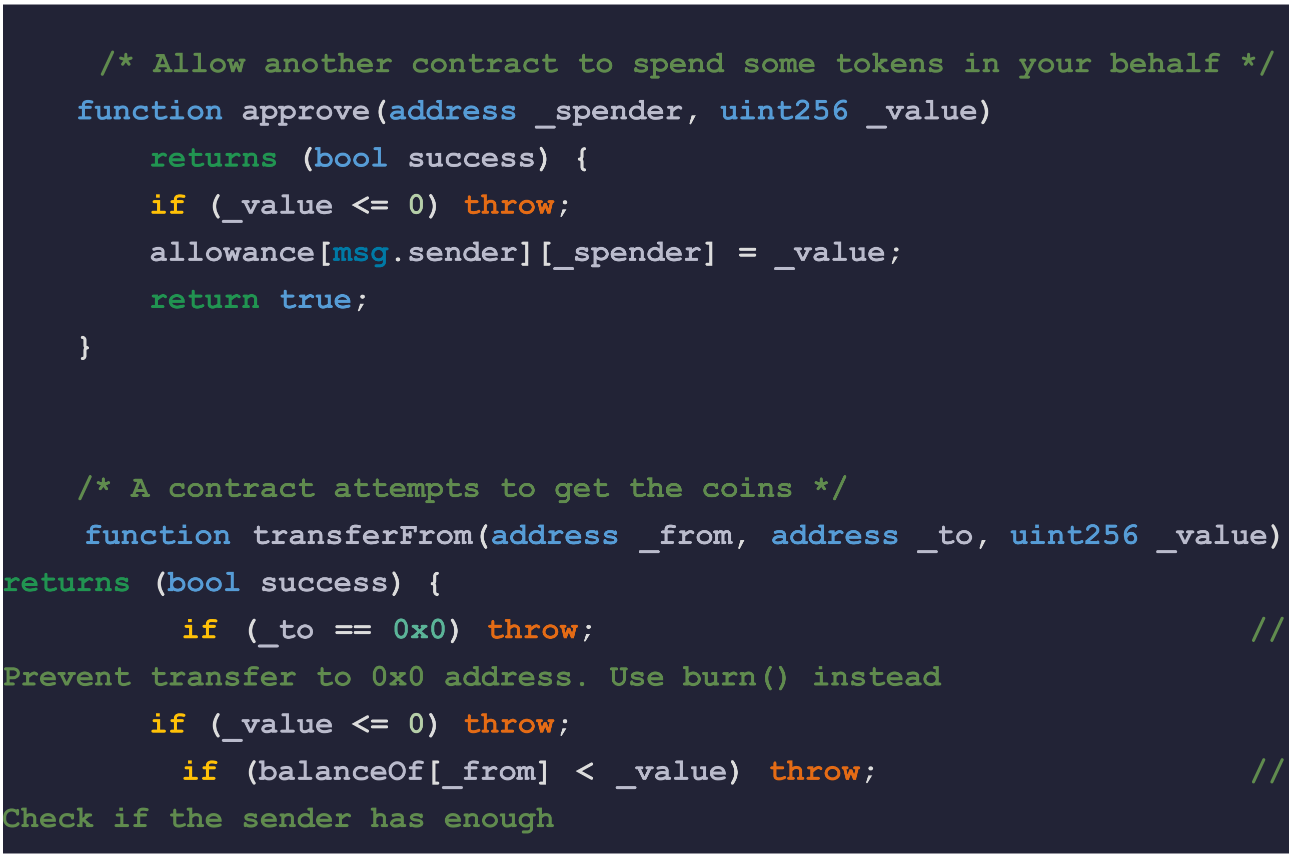
Resolution: Renounce the ownership if not needed anymore.

(3) No Rate Limiting or Lock Period for Freeze/Unfreeze:

There's no mechanism to prevent users from repeatedly freezing and unfreezing their tokens. Without rate limiting or lock periods, this could potentially be abused to spam transactions or disrupt the functioning of the contract.

Resolution: To mitigate abuse, implement rate limiting or lock periods for freezing and unfreezing tokens. Rate limiting restricts the frequency of these actions, while lock periods enforce a waiting period before tokens can be unfrozen. Additionally, consider permission-based systems or gradual unfreezing mechanisms to add further control and stability to the process.

(4) Gas optimization:



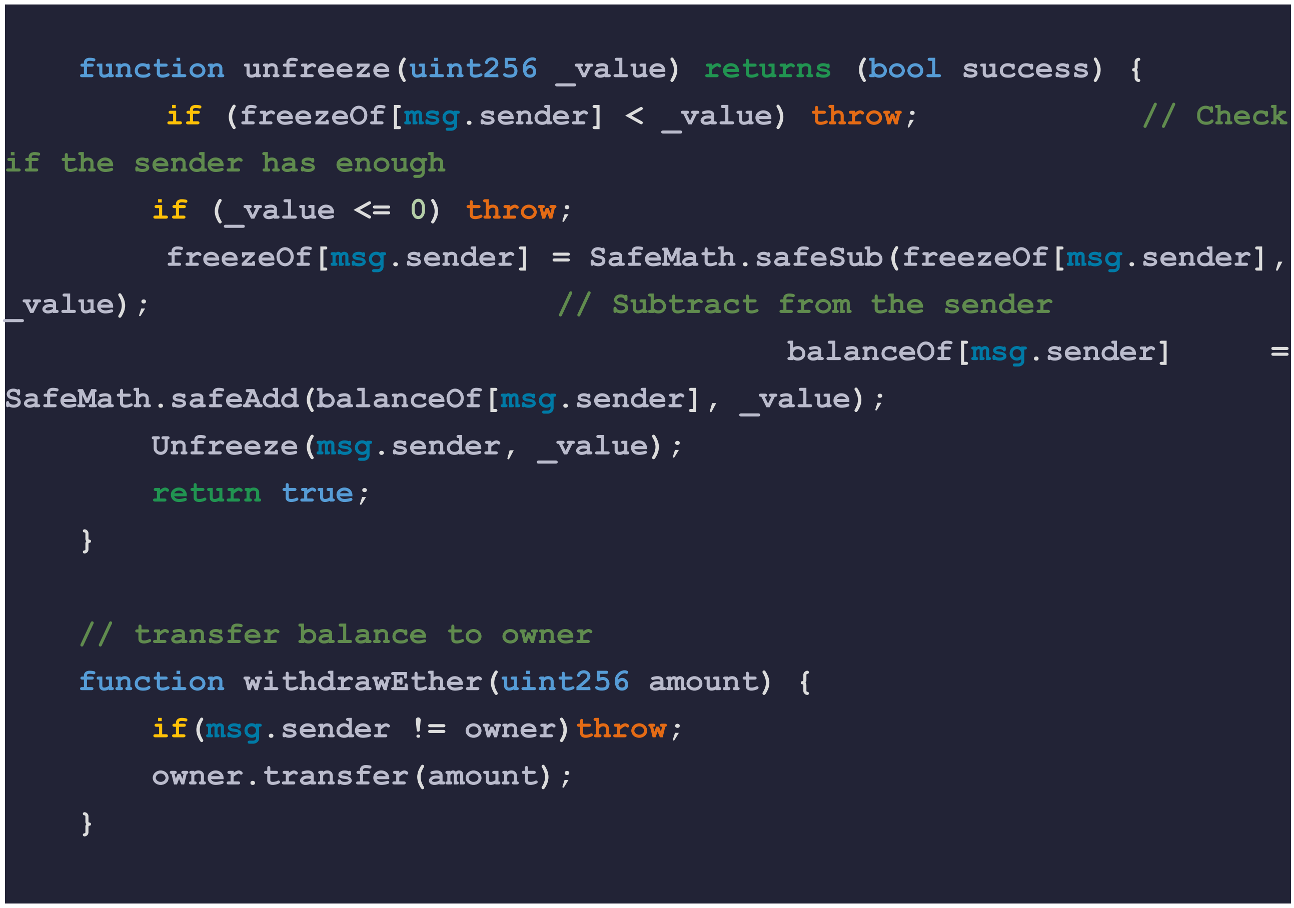
```
     /* Allow another contract to spend some tokens in your behalf */
    function approve(address _spender, uint256 _value)
        returns (bool success) {
        if (_value <= 0) throw;
        allowance[msg.sender][_spender] = _value;
        return true;
    }


    /* A contract attempts to get the coins */
    function transferFrom(address _from, address _to, uint256 _value)
returns (bool success) {
         if (_to == 0x0) throw;                                   //
Prevent transfer to 0x0 address. Use burn() instead
        if (_value <= 0) throw;
         if (balanceOf[_from] < _value) throw;                    //
Check if the sender has enough
```

```
            if  (balanceOf[_to] + _value < balanceOf[_to]) throw;   //
Check for overflows
            if  (_value > allowance[_from][msg.sender]) throw;      //
Check allowance
                balanceOf[_from]  = SafeMath.safeSub(balanceOf[_from],
_value);                                // Subtract from the sender
            balanceOf[_to] = SafeMath.safeAdd(balanceOf[_to], _value);
// Add the same to the recipient
                                    allowance[_from][msg.sender]    =
SafeMath.safeSub(allowance[_from][msg.sender], _value);
        Transfer(_from, _to, _value);
        return true;
    }

    function burn(uint256 _value) returns (bool success) {
          if  (balanceOf[msg.sender] < _value) throw;               //
Check if the sender has enough
        if (_value <= 0) throw;
                                          balanceOf[msg.sender]     =
SafeMath.safeSub(balanceOf[msg.sender],                       _value);
// Subtract from the sender
                 totalSupply  =  SafeMath.safeSub(totalSupply,_value);
// Updates totalSupply
        Burn(msg.sender, _value);
        return true;
    }

    function freeze(uint256 _value) returns (bool success) {
          if  (balanceOf[msg.sender] < _value) throw;               //
Check if the sender has enough
        if (_value <= 0) throw;
                                          balanceOf[msg.sender]     =
SafeMath.safeSub(balanceOf[msg.sender],                       _value);
// Subtract from the sender
         freezeOf[msg.sender] = SafeMath.safeAdd(freezeOf[msg.sender],
_value);                                    // Updates totalSupply
        Freeze(msg.sender, _value);
        return true;
    }
```



```
    function unfreeze(uint256 _value) returns (bool success) {
         if (freezeOf[msg.sender] < _value) throw;            // Check
if the sender has enough
        if (_value <= 0) throw;
         freezeOf[msg.sender] = SafeMath.safeSub(freezeOf[msg.sender],
_value);                              // Subtract from the sender
                                                 balanceOf[msg.sender]      =
SafeMath.safeAdd(balanceOf[msg.sender], _value);
        Unfreeze(msg.sender, _value);
        return true;
    }

    // transfer balance to owner
    function withdrawEther(uint256 amount) {
        if(msg.sender != owner)throw;
        owner.transfer(amount);
    }
```

The “public” functions which are never called by the contract internally, could be declared “external” to save the gas cost.

Resolution: The “external” functions are more efficient than “public” functions.

Centralization Risk

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

BNB.sol

- withdrawEther: Withdraw ether balance transfer to owner.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of Etherscan web links. And we have used all possible tests based on given objects as files. We had observed 4 informational issues in the smart contracts. And those issues are not critical. So, **it's good to go for the production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

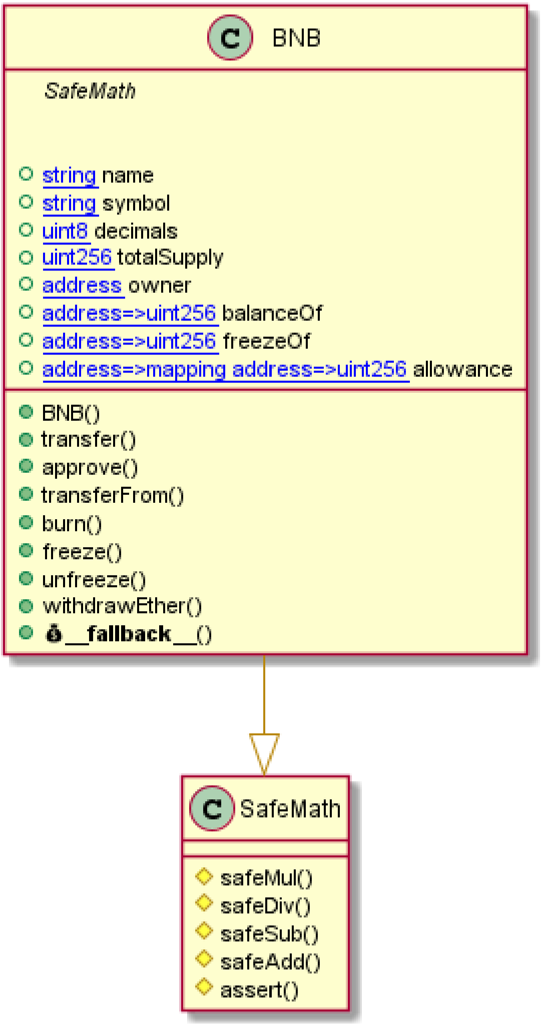
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - BNB Token

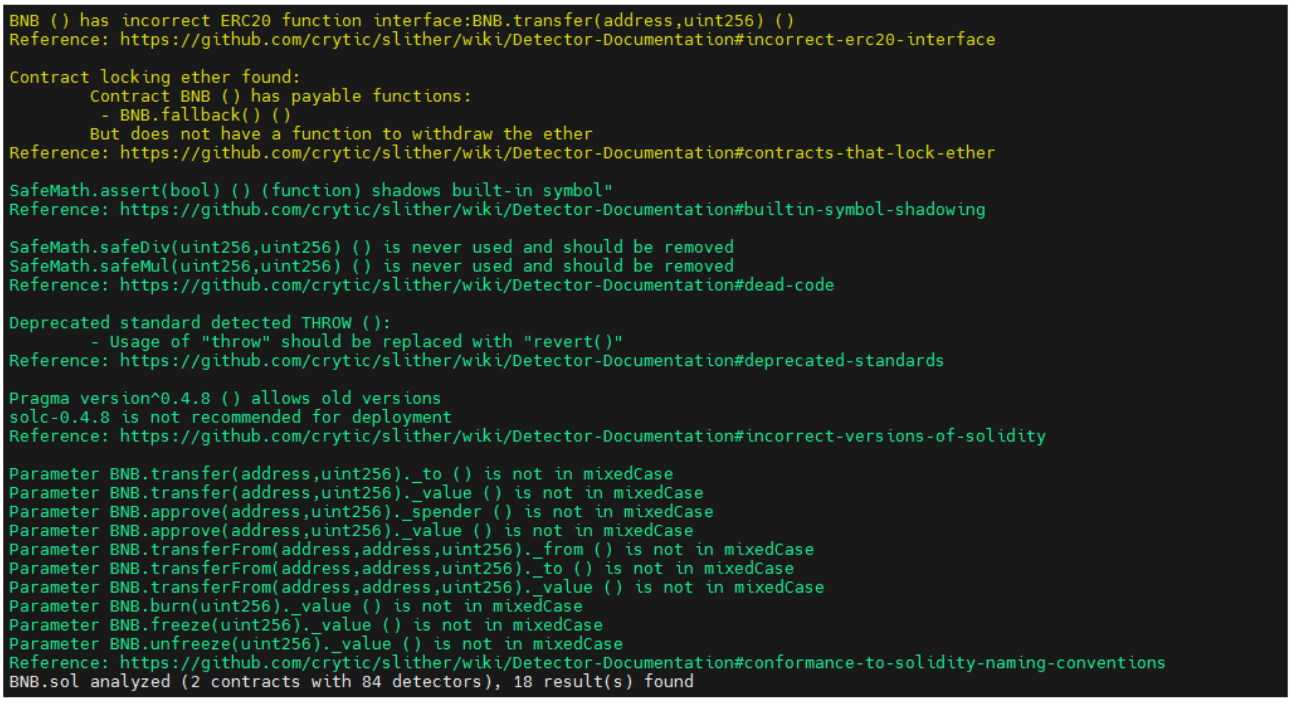


Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

Slither Log >> BNB.sol



```
BNB () has incorrect ERC20 function interface:BNB.transfer(address,uint256) ()
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-erc20-interface

Contract locking ether found:
        Contract BNB () has payable functions:
         - BNB.fallback() ()
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether

SafeMath.assert(bool) () (function) shadows built-in symbol"
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#builtin-symbol-shadowing

SafeMath.safeDiv(uint256,uint256) () is never used and should be removed
SafeMath.safeMul(uint256,uint256) () is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Deprecated standard detected THROW ():
        - Usage of "throw" should be replaced with "revert()"
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#deprecated-standards

Pragma version^0.4.8 () allows old versions
solc-0.4.8 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Parameter BNB.transfer(address,uint256)._to () is not in mixedCase
Parameter BNB.transfer(address,uint256)._value () is not in mixedCase
Parameter BNB.approve(address,uint256)._spender () is not in mixedCase
Parameter BNB.approve(address,uint256)._value () is not in mixedCase
Parameter BNB.transferFrom(address,address,uint256)._from () is not in mixedCase
Parameter BNB.transferFrom(address,address,uint256)._to () is not in mixedCase
Parameter BNB.transferFrom(address,address,uint256)._value () is not in mixedCase
Parameter BNB.burn(uint256)._value () is not in mixedCase
Parameter BNB.freeze(uint256)._value () is not in mixedCase
Parameter BNB.unfreeze(uint256)._value () is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
BNB.sol analyzed (2 contracts with 84 detectors), 18 result(s) found
```

Solidity Static Analysis

Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

BNB.sol

Software analysis result:

These software checked contracts but did not find any issues in the file.

Solhint Linter

Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

BNB.sol.

Software analysis result:

These software checked contracts but did not find any issues in the file.

EtherAuthority

Email: audit@EtherAuthority.io