SPHINXSHIELD

Security Assessment

BNB Whales

Oct 23th, 2023

Evaluation Outcomes

Security Score

Review	Score
Overall Score	83/100
Auditor Score	81/100

Review by Section	Score
Manual Scan Score	29/57
Advance Check Score	11/19

Scoring System

This scoring system is provided to gauge the overall value of the audit. The maximum achievable score is 100, but reaching this score requires the project to meet all assessment requirements.

Our updated passing score is now set at 80 points. If a project fails to achieve at least 80% of the total score, it will result in an automatic failure.

Please refer to our notes and final assessment for more details.

Table of Contents

Summary

This audit report is tailored for **BNB Whales**, aiming to uncover potential issues and vulnerabilities within the **BNB Whales** project's source code, along with scrutinizing contract dependencies outside recognized libraries. Our audit comprises a comprehensive investigation involving Static Analysis and Manual Review techniques.

Our audit process places a strong emphasis on the following focal points:

1. Rigorous testing of smart contracts against both commonplace and rare attack vectors.
2. Evaluation of the codebase for alignment with contemporary best practices and industry standards.
3. Ensuring the contract logic is in harmony with the client's specifications and objectives.
4. A comparative analysis of the contract structure and implementation against analogous smart contracts created by industry frontrunners.
5. An exhaustive, line-by-line manual review of the entire codebase by domain experts.

The outcome of this security assessment yielded findings spanning from critical to informational. To uphold robust security standards and align with industry norms, we present the following security-driven recommendations:

1. Elevate general coding practices to optimize source code structure.
2. Implement an all-encompassing suite of unit tests to account for all conceivable use cases.
3. Enhance codebase transparency through increased commenting, particularly in externally verifiable contracts.
4. Improve clarity regarding privileged activities upon the protocol's transition to a live state.

Overview

Project Summary

Project Name	BNB Whales
Blockchain	Binance Smart Chain
Language	Solidity
Codebase	https://bscscan.com/address/0x8689DeA88EC1E2638250Eef702E722C6dACFF70E
Commit	

Audit Summary

Delivery Date	Oct 23th, 2023
Audit Methodology	Static Analysis, Manual Review
Key Components	

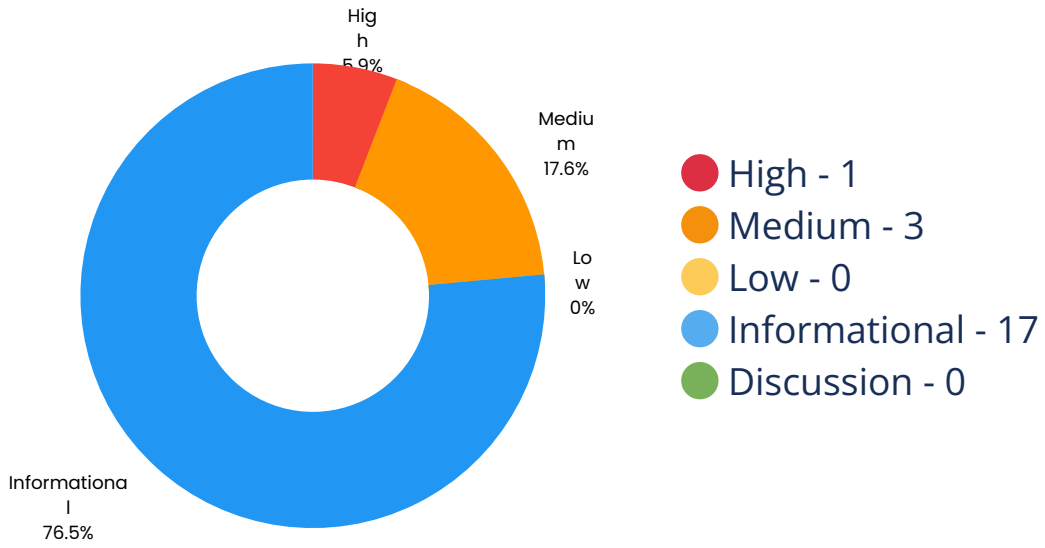
Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Aknowledged	Resolved
High	1	0	0	1	0
Medium	3	0	0	3	0
Low	0	0	0	0	0
Informational	17	0	0	17	0
Discussion	0	0	0	0	0

Audit Scope

ID	File	KECCAK256 or SHA256 Checksum
SPM	BNBWhales.sol	0xcd989df10a349b74e5a1f135f7e5835d124f6bef93c154dfe4d7445ad8744df2

Findings



Location	Title	Scope	Severity	Status
BNBWhales.sol:545	Freeze Money	BNBWhales	High	Aknowledged
BNBWhales.sol:672	Unexpected Ether Balance	BNBWhales	Medium	Aknowledged
BNBWhales.sol:760	Unexpected Ether Balance	BNBWhales	Medium	Aknowledged
BNBWhales.sol:762	Unexpected Ether Balance	BNBWhales	Medium	Aknowledged
BNBWhales.sol:339	Prefer uint256	BNBWhales	Informational	Aknowledged
BNBWhales.sol:16, 141,48,325,3	Recommend to Follow Code Layout Conventions	Ownable	Informational	Aknowledged
BNBWhales.sol:825 ,532,473,459,523	No Check of Address Params with Zero Address	BNBWhales	Informational	Aknowledged
BNBWhales.sol:337 ,366,339,338,342	Variables Should Be Constants	BNBWhales	Informational	Aknowledged
BNBWhales.sol:714	Use Shift Operation Instead of Mul/Div	BNBWhales	Informational	Aknowledged

Location	Title	Scope	Severity	Status
BNBWhales.sol:677	Continuous State Variable Write	BNBWhales	Informational	Aknowledged
BNBWhales.sol:583,534	Use ++i/--i Instead of i++/i--	BNBWhales	Informational	Aknowledged
BNBWhales.sol:598,613,596,617,641,625,430,582,525,783,633,784,429,608,760,606,621,767,549,419,548,581	Cache State Variables that are Read Multiple Times within A Function	BNBWhales	Informational	Aknowledged
BNBWhales.sol:667	Use != 0 Instead of > 0 for Unsigned Integer Comparison	BNBWhales	Informational	Aknowledged
BNBWhales.sol:484,442,648,479,473,464,493,523,446,506,497,438,489,459,468	Function Visibility Can Be External	BNBWhales	Informational	Aknowledged
BNBWhales.sol:393,524,533,744,666,826,499,507,518,667,32,42,654,653	Use CustomError Instead of String	Ownable	Informational	Aknowledged
BNBWhales.sol:393	Lack of Error Message	BNBWhales	Informational	Aknowledged
BNBWhales.sol:373	Unused State Variables	BNBWhales	Informational	Aknowledged
BNBWhales.sol:518,744,666,667,826,42,654,653,499	Long String in revert/require	Ownable	Informational	Aknowledged
BNBWhales.sol:353,350,360,362,364,349,352,368,359,346,347,363,369	Variables Can Be Declared as Immutable	BNBWhales	Informational	Aknowledged

Location	Title	Scope	Severity	Status
BNBWhales.sol:736,700,723,713	Get Contract Balance of ETH in Assembly	BNBWhales	Informational	Aknowledged
BNBWhales.sol:666,42,654,653	Use Assembly to Check Zero Address	Ownable	Informational	Aknowledged

Code Security - Freeze Money

Title	Severity	Location	Status
Freeze Money	High	BNBWhales.sol:545	Aknowledged

Description

There is at least one payable function in the contract, but no transfer function(like send, transfer, call...) exists, which will cause Ether to be locked in the contract.

Code Security - Unexpected Ether Balance

Title	Severity	Location	Status
Unexpected Ether Balance	Medium	BNBWhales.sol:672	Aknowledged

Description

Strict checks on the balance of the account may cause the contract to run abnormally, because the change of the balance of the account may be affected by various factors. For example, the selfdestruct method in a contract allows sending arbitrary Ether to another contract without triggering the fallback function of another contract.

Code Security - Unexpected Ether Balance

Title	Severity	Location	Status
Unexpected Ether Balance	Medium	BNBWhales.sol:760	Aknowledged

Description

Strict checks on the balance of the account may cause the contract to run abnormally, because the change of the balance of the account may be affected by various factors. For example, the selfdestruct method in a contract allows sending arbitrary Ether to another contract without triggering the fallback function of another contract.

Code Security - Unexpected Ether Balance

Title	Severity	Location	Status
Unexpected Ether Balance	Medium	BNBWhales.sol:762	Aknowledged

Description

Strict checks on the balance of the account may cause the contract to run abnormally, because the change of the balance of the account may be affected by various factors. For example, the selfdestruct method in a contract allows sending arbitrary Ether to another contract without triggering the fallback function of another contract.

Optimization Suggestion - Prefer uint256

Title	Severity	Location	Status
Prefer uint256	Informational	BNBWhales.sol:339	Aknowledged

Description

It is recommended to use uint256/int256 types to avoid gas overhead caused by 32 bytes padding.

Optimization Suggestion - Recommend to Follow Code Layout Conventions

Title	Severity	Location	Status
Recommend to Follow Code Layout Conventions	Informational	BNBWhales.sol:16,141 ,48,325,3	Aknowledged

Description

In the solidity document(https://docs.soliditylang.org/en/v0.8.17/style-guide.html), there are the following conventions for code layout: Layout contract elements in the following order: 1. Pragma statements, 2. Import statements, 3. Interfaces, 4. Libraries, 5. Contracts. Inside each contract, library or interface, use the following order: 1. Type declarations, 2. State variables, 3. Events, 4. Modifiers, 5. Functions. Functions should be grouped according to their visibility and ordered: 1. constructor, 2. receive function (if exists), 3. fallback function (if exists), 4. external, 5. public, 6. internal, 7. private.

Optimization Suggestion - No Check of Address Params with Zero Address

Title	Severity	Location	Status
No Check of Address Params with Zero Address	Informational	BNBWhales.sol:825,532,473,459,523	Aknowledged

Description

The input parameter of the address type in the function does not use the zero address for verification.

Optimization Suggestion - Variables Should Be Constants

Title	Severity	Location	Status
Variables Should Be Constants	Informational	BNBWhales.sol:337,366,339,338,342	Aknowledged

Description

There are unchanging state variables in the contract, and putting unchanging state variables in storage will waste gas.

Optimization Suggestion - Use Shift Operation Instead of Mul/Div

Title	Severity	Location	Status
Use Shift Operation Instead of Mul/Div	Informational	BNBWhales.sol:714	Aknowledged

Description

It is recommended to use shift operation instead of direct multiplication and division if possible, because shift operation is more gas-efficient.

Optimization Suggestion - Continuous State Variable Write

Title	Severity	Location	Status
Continuous State Variable Write	Informational	BNBWhales.sol:677	Aknowledged

Description

When there are multiple continuous write operations on a state variable, the intermediate write operations are redundant and will cost more gas.

Optimization Suggestion - Use ++i/--i Instead of i++/i--

Title	Severity	Location	Status
Use ++i/--i Instead of i++/i--	Informational	BNBWhales.sol:583,534	Aknowledged

Description

Compared with i++, ++i can save about 5 gas per use. Compared with i--, --i can save about 3 gas per use in for loop.

Optimization Suggestion - Cache State Variables that are Read Multiple Times within A Function

Title	Severity	Location	Status
Cache State Variables that are Read Multiple Times within A Function	Informational	BNBWhales.sol:598,613,596,617,641,625,430,582,525,783,633,784,429,608,760,606,621,767,549,419,548,581	Aknowledged

Description

When a state variable is read multiple times in a function, using a local variable to cache the state variable can avoid frequently reading data from storage, thereby saving gas.

Optimization Suggestion - Use != 0 Instead of > 0 for Unsigned Integer Comparison

Title	Severity	Location	Status
Use != 0 Instead of > 0 for Unsigned Integer Comparison	Informational	BNBWhales.sol:667	Aknowledged

Description

For unsigned integers, use !=0 for comparison, which consumes less gas than >0. When compiler optimization is turned off, about 3 gas can be saved. When compiler optimization is turned on, no gas can be saved.

Optimization Suggestion - Function Visibility Can Be External

Title	Severity	Location	Status
Function Visibility Can Be External	Informational	BNBWhales.sol:484,442,648,479,473,464,493,523,446,506,497,438,489,459,468	Aknowledged

Description

Functions that are not called should be declared as external.

Optimization Suggestion - Use CustomError Instead of String

Title	Severity	Location	Status
Use CustomError Instead of String	Informational	BNBWhales.sol:393,524,533,744,666,826,499,507,518,667,32,42,654,653	Aknowledged

Description

When using require or revert, CustomError is more gas efficient than string description, as the error message described using CustomError is only compiled into four bytes. Especially when string exceeds 32 bytes, more gas will be consumed. Generally, around 250-270 gas can be saved for one CustomError replacement when compiler optimization is turned off, 60-80 gas can be saved even if compiler optimization is turned on.

Optimization Suggestion - Lack of Error Message

Title	Severity	Location	Status
Lack of Error Message	Informational	BNBWhales.sol:393	Aknowledged

Description

Use empty string as parameter while invoking function Revert() or Require().

Optimization Suggestion - Unused State Variables

Title	Severity	Location	Status
Unused State Variables	Informational	BNBWhales.sol:373	Aknowledged

Description

The state variables are not used, which will increase the gas consumption.

Optimization Suggestion - Long String in revert/require

Title	Severity	Location	Status
Long String in revert/require	Informational	BNBWhales.sol:518,744,666,667,826,42,654,653,499	Aknowledged

Description

If the string parameter in the revert/require function exceeds 32 bytes, more gas will be consumed.

Optimization Suggestion - Variables Can Be Declared as Immutable

Title	Severity	Location	Status
Variables Can Be Declared as Immutable	Informational	BNBWhales.sol:353,350,360,362,364,349,352,368,359,346,347,363,369	Aknowledged

Description

The solidity compiler of version 0.6.5 introduces immutable to modify state variables that are only modified in the constructor. Using immutable can save gas.

Optimization Suggestion - Get Contract Balance of ETH in Assembly

Title	Severity	Location	Status
Get Contract Balance of ETH in Assembly	Informational	BNBWhales.sol:736,700,723,713	Aknowledged

Description

Using the selfbalance and balance opcodes to get the ETH balance of the contract in assembly saves gas compared to getting the ETH balance through address(this).balance and xx.balance. When compiler optimization is turned off, about 210-250 gas can be saved, and when compiler optimization is turned on, about 50-100 gas can be saved.

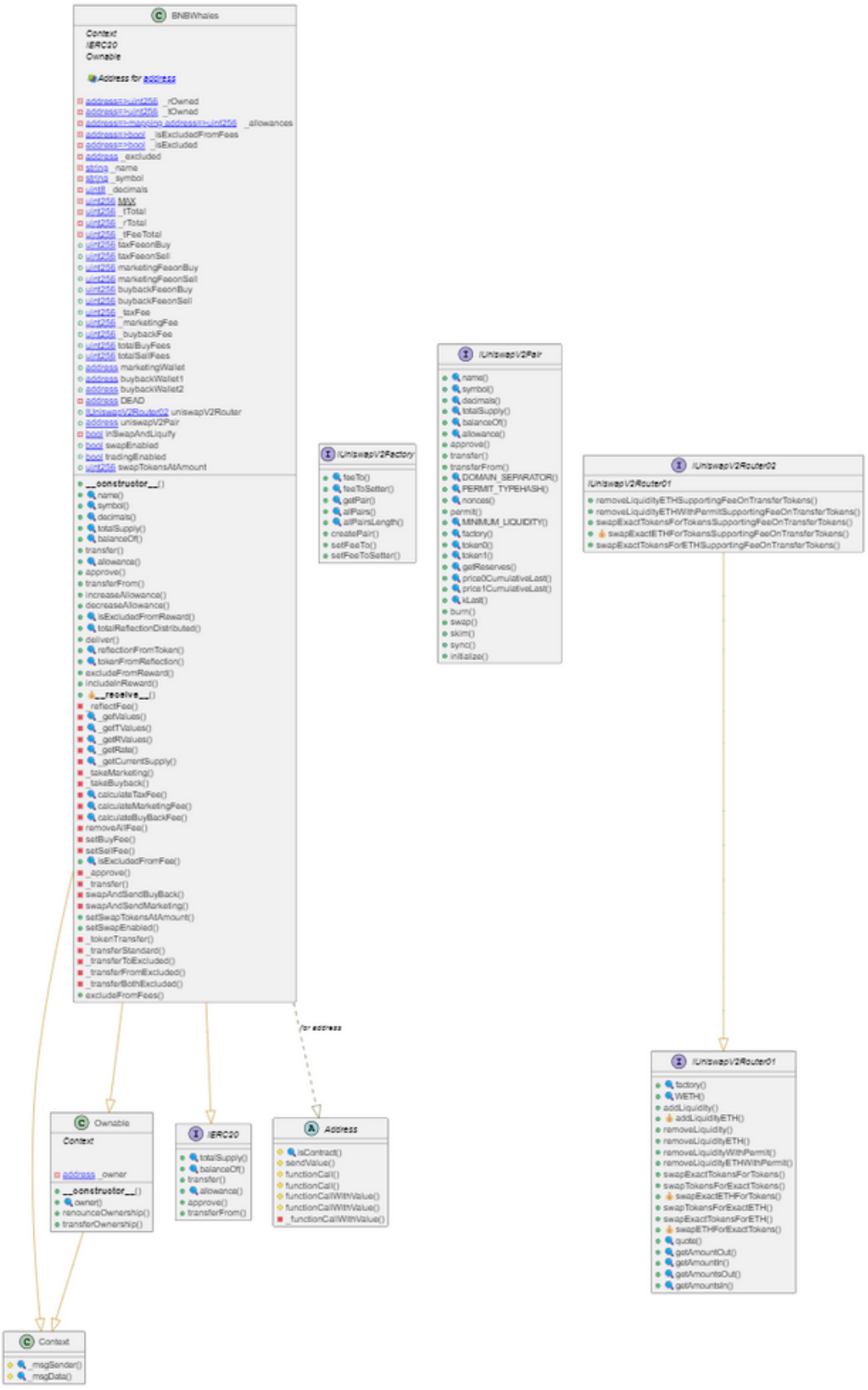
Optimization Suggestion - Use Assembly to Check Zero Address

Title	Severity	Location	Status
Use Assembly to Check Zero Address	Informational	BNBWhales.sol:666,42,654,653	Aknowledged

Description

Using assembly to check zero address can save gas. About 18 gas can be saved in each call.

PlantUML


BNBWhales
Context
IERC20
Ownable
Address for address
__constructor__()
name()
symbol()
decimals()
totalSupply()
balanceOf()
transfer()
allowance()
approve()
transferFrom()
increaseAllowance()
decreaseAllowance()
isExcludedFromReward()
totalReflectionDistributed()
deliver()
reflectionFromToken()
tokenFromReflection()
excludeFromReward()
includeInReward()
__receive__()
_reflectFee()
_getValues()
_getTValues()
_getRValues()
_getRate()
_getCurrentSupply()
_takeMarketing()
_takeBuyback()
calculateTaxFee()
calculateMarketingFee()
calculateBuyBackFee()
removeAllFee()
setBuyFee()
setSellFee()
isExcludedFromFee()
_approve()
_transfer()
swapAndSendBuyBack()
swapAndSendMarketing()
setSwapTokensAtAmount()
setSwapEnabled()
_tokenTransfer()
_transferStandard()
_transferToExcluded()
_transferFromExcluded()
_transferBothExcluded()
excludeFromFees()
for address
IUniswapV2Factory
feeTo()
feeToSetter()
getPair()
allPairs()
allPairsLength()
createPair()
setFeeTo()
setFeeToSetter()
IUniswapV2Pair
name()
symbol()
decimals()
totalSupply()
balanceOf()
allowance()
approve()
transfer()
transferFrom()
DOMAIN_SEPARATOR()
PERMIT_TYPEHASH()
nonces()
permit()
MINIMUM_LIQUIDITY()
factory()
token0()
token1()
getReserves()
price0CumulativeLast()
price1CumulativeLast()
kLast()
burn()
swap()
skim()
sync()
initialize()
IUniswapV2Router02
IUniswapV2Router01
removeLiquidityETHSupportingFeeOnTransferTokens()
removeLiquidityETHWithPermitSupportingFeeOnTransferTokens()
swapExactTokensForTokensSupportingFeeOnTransferTokens()
swapExactETHForTokensSupportingFeeOnTransferTokens()
swapExactTokensForETHSupportingFeeOnTransferTokens()
IUniswapV2Router01
factory()
WETH()
addLiquidity()
addLiquidityETH()
removeLiquidity()
removeLiquidityETH()
removeLiquidityWithPermit()
removeLiquidityETHWithPermit()
swapExactTokensForTokens()
swapTokensForExactTokens()
swapExactETHForTokens()
swapTokensForExactETH()
swapExactTokensForETH()
swapETHForExactTokens()
quote()
getAmountOut()
getAmountIn()
getAmountsOut()
getAmountsIn()
Ownable
Context
address _owner
__constructor__()
owner()
renounceOwnership()
transferOwnership()
IERC20
totalSupply()
balanceOf()
transfer()
allowance()
approve()
transferFrom()
Address
isContract()
sendValue()
functionCall()
functionCall()
functionCallWithValue()
functionCallWithValue()
_functionCallWithValue()
Context
_msgSender()
_msgData()

Appendix

Finding Categories

Security and Best Practices

1. Freeze Money: Ensure that contracts are designed to handle the freezing of funds effectively to prevent unauthorized transfers or withdrawals.
2. Unexpected Ether Balance: Verify that Ether balances are managed correctly, and unexpected Ether balances are avoided.
3. Prefer uint256: Emphasize the use of the uint256 data type for integer variables to prevent potential overflow or underflow issues.
4. Recommend to Follow Code Layout Conventions: Enforce consistent code layout conventions to enhance code readability and maintainability.
5. No Check of Address Params with Zero Address: Validate address parameters to ensure they are not the zero address, preventing potential issues.
6. Variables Should Be Constants: Declare variables as constants if they are not intended to change during contract execution.
7. Use Shift Operation Instead of Mul/Div: Utilize bit-wise shift operations for multiplication and division operations to improve efficiency and reduce gas costs.
8. Continuous State Variable Write: Minimize state variable writes to enhance contract efficiency and reduce gas consumption.
9. Use ++i/--i Instead of i++/i--: Employ the ++i and --i increment and decrement operators over the i++ and i-- operators to prevent potential issues.
10. Cache State Variables that are Read Multiple Times within A Function: Cache frequently accessed state variables to reduce redundant read operations.
11. Use != 0 Instead of > 0 for Unsigned Integer Comparison: Use the != operator for comparisons with zero to prevent potential issues related to signed and unsigned integers.
12. Function Visibility Can Be External: Set function visibility to external if they are only accessed externally to enhance gas efficiency.
13. Use CustomError Instead of String: Prefer custom error codes over string error messages for improved contract efficiency.
14. Lack of Error Message: Include informative error messages to aid in debugging and error handling.
15. Unused State Variables: Avoid unused state variables, as they can lead to unnecessary gas costs and confusion in code.
16. Long String in revert/require: Optimize long revert or require strings for efficient gas usage.
17. Variables Can Be Declared as Immutable: Declare variables as immutable if their values will not change after initialization to enhance security and readability.
18. Get Contract Balance of ETH in Assembly: Employ optimized assembly checks to verify contract balances efficiently.

KECCAK256 or SHA256 Checksum Verification

Checksum verification is a critical component of smart contract development. It ensures the integrity of contract deployment and code execution by confirming that the bytecode being executed matches the intended source code. The following details the KECCAK256 and SHA256 checksum verification process.

KECCAK256 Checksum Verification:

- Checksum Definition: KECCAK256 is a cryptographic hashing function used in Ethereum to create a checksum of the contract bytecode. It is part of the Ethereum Name Service (ENS) standard.
- Use Cases: KECCAK256 checksums are used in ENS for verification of Ethereum addresses. They help prevent unintended transfers due to typos or errors.
- Checksum Process: The KECCAK256 checksum is created by taking the SHA3 hash of the lowercase hexadecimal Ethereum address, and then converting it to the corresponding checksum address by replacing characters with uppercase letters.

SHA256 Checksum Verification:

- Checksum Definition: SHA256 is a widely used cryptographic hash function, often employed to verify the integrity of data and contracts.
- Use Cases: SHA256 checksums are widely used in software development, including the verification of software downloads and smart contracts.
- Checksum Process: The SHA256 checksum is generated by applying the SHA256 hashing algorithm to the content of the contract. This results in a fixed-length hexadecimal value that is compared to the expected value to verify the contract's integrity.

Importance of Checksum Verification:

- Checksum verification ensures that smart contracts are executed as intended, preventing tampering and security vulnerabilities.
- It is a security best practice to verify that the deployed bytecode matches the intended source code, reducing the risk of unexpected behavior.

Best Practices:

- Always use checksum verification in situations where it is essential to verify Ethereum addresses or contract integrity.
- Implement checksum verification to ensure that contract deployment and interactions occur as intended.
- Verify the validity of contract deployments and the integrity of the code during development and deployment phases.

Website Scan

https://bnbwhales.net/

Network Security

High | 0 Attentions

Application Security

High | 2 Attentions

DNS Security

High | 0 Attentions

Network Security

9 Passed **0 Attention**

FTP Service Anonymous LOGIN	NO
VNC Service Accesible	NO
RDP Service Accesible	NO
LDAP Service Accesible	NO
PPTP Service Accesible	NO
RSYNC Service Accesible	NO
SSH Weak Cipher	NO
SSH Support Weak MAC	NO
CVE on the Related Service	NO

Application Security

9 Passed | 2 Attention

Missing X-Frame-Options Header	NO
Missing HSTS header	YES
Missing X-Content-Type-Options Header	NO
Missing Content Security Policy (CSP)	YES
HTTP Access Allowed	NO
Self-Signed Certificate	NO
Wrong Host Certificate	NO
Expired Certificate	NO
SSL/TLS Supports Weak Cipher	NO
Support SSL Protocols	NO
Support TLS Weak Version	NO

DNS Health

10 Passed | 0 Attention

Missing SPF Record	NO
Missing DMARC Record	NO
Missing DKIM Record	NO
Ineffective SPF Record	NO
SPF Record Contains a Softfail Without DMARC	NO
Name Servers Versions Exposed	NO
Allow Recursive Queries	NO
CNAME in NS Records	NO
MX Records IPs are Private	NO
MX Records has Invalid Chars	NO

Social Media Checks

2 Passed | 7 Failed

X (Twitter)		PASS
Facebook		FAIL
Instagram		FAIL
TikTok		FAIL
YouTube		FAIL
Twich		FAIL
Telegram		PASS
Discord		FAIL
Others		FAIL

Recommendation

To enhance project credibility and outreach, we suggest having a minimum of three active social media channels and a fully functional website.

Social Media Information Notes

Unspecified Auditor Notes

Notes from the Project Owner

Fundamental Health

KYC Status

SphinxShield KYC	NO
3rd Party KYC	NO

Project Maturity Metrics

Minimally Developed	LOW
Token Launch Date	2023.10.24 14:30 (UTC)
Token Market Cap (estimate)	$13,021
Token/Project Age	NOT AVAILABLE

Recommendation

We strongly recommend that the project undergo the Know Your Customer (KYC) verification process with SphinxShield to enhance transparency and build trust within the crypto community. Furthermore, we encourage the project team to reach out to us promptly to rectify any inaccuracies or discrepancies in the provided information to ensure the accuracy and reliability of their project data.

Disclaimer

SphinxShield, its agents, and associates provide the information and content contained within this audit report and materials (collectively referred to as "the Services") for informational and security assessment purposes only. The Services are provided "as is" without any warranty or representation of any kind, either express or implied. SphinxShield, its agents, and associates make no warranty or undertaking, and do not represent that the Services will:

- Meet customer's specific requirements.
- Achieve any intended results.
- Be compatible or work with any other software, applications, systems, or services.
- Operate without interruption.
- Meet any performance or reliability standards.
- Be error-free or that any errors or defects can or will be corrected.

In addition, SphinxShield, its agents, and associates make no representation or warranty of any kind, express or implied, as to the accuracy, reliability, or currency of any information or content provided through the Services. SphinxShield assumes no liability or responsibility for any:

- Errors, mistakes, or inaccuracies of content and materials.
- Loss or damage of any kind incurred as a result of the use of any content.
- Personal injury or property damage, of any nature whatsoever, resulting from customer's access to or use of the Services, assessment report, or other materials.

It is imperative to recognize that the Services, including any associated assessment reports or materials, should not be considered or relied upon as any form of financial, tax, legal, regulatory, or other advice.

SphinxShield provides the Services solely to the customer and for the purposes specifically identified in this agreement. The Services, assessment reports, and accompanying materials may not be relied upon by any other person or for any purpose not explicitly stated in this agreement. Copies of these materials may not be delivered to any other person without SphinxShield's prior written consent in each instance.

Furthermore, no third party or anyone acting on behalf of a third party shall be considered a third-party beneficiary of the Services, assessment reports, and any accompanying materials. No such third party shall have any rights of contribution against SphinxShield with respect to the Services, assessment reports, and any accompanying materials.

The representations and warranties of SphinxShield contained in this agreement are solely for the benefit of the customer. Accordingly, no third party or anyone acting on behalf of a third party shall be considered a third-party beneficiary of such representations and warranties. No such third party shall have any rights of contribution against SphinxShield with respect to such representations or warranties or any matter subject to or resulting in indemnification under this agreement or otherwise.

About

SphinxShield, established in 2023, is a cybersecurity and auditing firm dedicated to fortifying blockchain and cryptocurrency security. We specialize in providing comprehensive security audits and solutions, aimed at protecting digital assets and fostering a secure investment environment.

Our accomplished team of experts possesses in-depth expertise in the blockchain space, ensuring our clients receive meticulous code audits, vulnerability assessments, and expert security advice. We employ the latest industry standards and innovative auditing techniques to reveal potential vulnerabilities, guaranteeing the protection of our clients' digital assets against emerging threats.

At SphinxShield, our unwavering mission is to promote transparency, security, and compliance with industry standards, contributing to the growth of blockchain and cryptocurrency projects. As a forward-thinking company, we remain adaptable, staying current with emerging trends and technologies to consistently enhance our services.

SphinxShield is your trusted partner for securing crypto ventures, empowering you to explore the vast potential of blockchain technology with confidence.