

# SPHINXSHIELD

## Security Assessment

### **BNB AI**

### Nov 10th, 2023

Disclaimer: SphinxShield conducts security assessments on the provided source code exclusively.  
Conduct your own due diligence before deciding to use any info listed at this page.



# Evaluation Outcomes

## Security Score

Review	Score
Overall Score	89/100
Auditor Score	84/100

Review by Section	Score
Manual Scan Score	47/57
Advance Check Score	16/19

## 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**

### **Overview**

[Project Summary](#)

[Audit Summary](#)

[Vulnerability Summary](#)

[Audit Scope](#)

### **Understandings**

### **Findings**

### **PlantUML**

### **Appendix**

### **Website Scan**

### **Social Media Checks**

### **Fundamental Health**

### **Coin Tracker Analytics**

### **CEX Holding Analytics**

### **Disclaimer**

### **About**



# Summary

This audit report is tailored for **BNB AI**, aiming to uncover potential issues and vulnerabilities within the **BNB AI** 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 AI
Blockchain	Binance Smart Chain
Language	Solidity
Codebase	<a href="https://bscscan.com/address/0x5566354c13A63ADB113457d804Dd63469a2a6B96">https://bscscan.com/address/0x5566354c13A63ADB113457d804Dd63469a2a6B96</a>
Commit	2db260bc803fcb8fa19c3728213f3f403e842936e9f0beba48a1b4ed52dede0d

## Audit Summary

Delivery Date	Nov 10th, 2023
Audit Methodology	Static Analysis, Manual Review
Key Components	BNBAI.sol

## Vulnerability Summary



Vulnerability Level	Total	⚠ Pending	⊗ Declined	ℹ Acknowledged	✓ Resolved
High	0	0	0	0	0
Medium	1	0	0	1	0
Low	2	0	0	2	0
Informational	19	0	0	19	0
Discussion	0	0	0	0	0



# Audit Scope

ID	File	KECCAK256 or SHA256 Checksum
BNA	BNBAI.sol	0x788b3fb99ff6a8040fc5319a9bef95a63aeb2a6cd0b87cbf0850a257d456610c



# Understandings

BNBAI is a BEP-20 token deployed on the Binance Smart Chain (BSC) blockchain.

## Token Information

- Token Name: BNB AI
- Symbol: BNBAI
- Decimals: 9
- Total Supply: 420,000,000,000 BNBAI

## Tax Distribution

Transactions involving BNBAI tokens incur a total fee, which is distributed across various components:

- RFI Fee: This fee contributes to the reflection mechanism, rewarding holders. The percentage is adjustable by the owner.
- Marketing Fee: A portion of the fee is allocated to marketing efforts to support the project's growth. The marketing fee is configurable by the owner.
- Ops Fee: Allocated for operational purposes, this fee supports the day-to-day functions of the project. The ops fee is adjustable by the owner.
- Liquidity Fee: This fee is reserved for providing liquidity to the token. The value is set by the owner.
- Dev Fee: Allocated for development purposes, this fee supports ongoing project enhancements. The value can be adjusted by the owner.

## Fee Management

The contract provides the owner with the capability to manage various fees:

- Set Tax: The owner can configure the RFI fee, marketing fee, ops fee, liquidity fee, and dev fee, along with the fee denominator. These adjustments offer flexibility in managing the fee structure.
- Set Fee Multipliers: Fee percentages for buy, sell, and transfer transactions can be adjusted by the owner using multipliers.
- Set Fee Receivers: The addresses that receive various fee components, including auto-liquidity, marketing, dev, and ops fees, can be set by the owner.



## **Tax Exemption**

The contract empowers the owner to exempt specific addresses from fees, providing a mechanism for fee exemption to certain addresses. This feature is valuable for whitelisting specific wallets or contracts.

## **Ownership and Authorization**

The contract owner has the authority to authorize specific addresses, allowing them to access privileged functions. These functions are restricted by the `onlyOwner` modifier and are used for configuring the contract and address attributes.

## **Transaction Limits**

The contract enforces transaction limits to prevent excessive token movement. This measure ensures that users do not make transactions that exceed the defined limits, promoting fair and controlled token transfers.

## **Swap Mechanism**

BNBAI employs a swap mechanism to manage liquidity. When a predefined threshold of tokens is reached, a portion of the contract's balance is swapped to BNB via the PancakeSwap Router. This swap action may temporarily affect the token's price. The remaining balance is then supplied to the BNBAI-BNB liquidity pool.

## **Open Trading**

Trading can be restricted based on conditions defined by the owner. This ensures that trading remains closed until specific requirements are met, providing a level of control over the market activity.

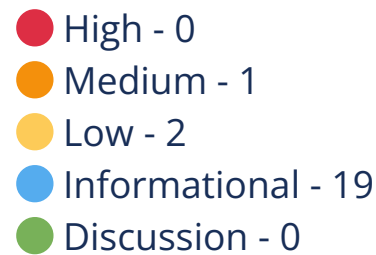
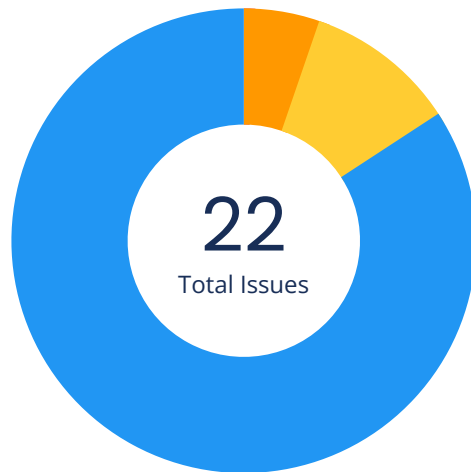
## **Additional Functionality**

The contract includes various additional functions, such as clearing stuck ETH, clearing tokens, and more. These functions contribute to the overall flexibility and utility of the BNBAI contract.





# Findings



Location	Title	Scope	Severity	Status
BNBAI.sol:738	Unchecked Call Return Value	BNBAI	Medium	Acknowledged
BNBAI.sol:732,738	Use Safer Functions	BNBAI	Low	Acknowledged
BNBAI.sol:478,508	Clarify Return Value	BNBAI	Low	Acknowledged
BNBAI.sol:732	Prefer .call() To send()/transfer()	BNBAI	Informational	Acknowledged
BNBAI.sol:5,38,115	Recommend to Follow Code Layout Conventions	IBEP20	Informational	Acknowledged
BNBAI.sol:698	Parameters Should Be Declared as Calldata	BNBAI	Informational	Acknowledged
BNBAI.sol:202,331,340,353,357,736	No Check of Address Params with Zero Address	BNBAI	Informational	Acknowledged
BNBAI.sol:139,147	Variables Should Be Constants	BNBAI	Informational	Acknowledged
BNBAI.sol:624,649,654,659	Use Shift Operation Instead of Mul/Div	BNBAI	Informational	Acknowledged



Location	Title	Scope	Severity	Status
BNBAI.sol:245,325,332,333,440,441,442,522,523,558,562,564,567,651,656,661,667,675,686	Cache State Variables that are Read Multiple Times within A Function	BNBAI	● Informational	Acknowledged
BNBAI.sol:342,524,699	Use ++i/--i Instead of i++/i--	BNBAI	● Informational	Acknowledged
BNBAI.sol:552	Use != 0 Instead of > 0 for Unsigned Integer Comparison	BNBAI	● Informational	Acknowledged
BNBAI.sol:226,230,234,239,248,252,257,271,276,287,292,296,331,353,357,361,736	Function Visibility Can Be External	BNBAI	● Informational	Acknowledged
BNBAI.sol:3	Floating Pragma	Global	● Informational	Acknowledged
BNBAI.sol:426,437,456,493,513	Optimizable Return Statement	BNBAI	● Informational	Acknowledged
BNBAI.sol:52,61,265,281,301,312,319,320,325,332,341,539,540,550,551,552,553,559,705,710,716,721,731,737	Use CustomError Instead of String	Ownable	● Informational	Acknowledged
BNBAI.sol:623	ReentrancyGuard Should Modify External Function	BNBAI	● Informational	Acknowledged
BNBAI.sol:61,265,281,319,320,325,539,540,550,551,552,553,705,710,716,721,737	Long String in revert/require	Ownable	● Informational	Acknowledged
BNBAI.sol:133,134	Variables Can Be Declared as Immutable	BNBAI	● Informational	Acknowledged



Location	Title	Scope	Severity	Status
BNBAI.sol:636,640,731	Get Contract Balance of ETH in Assembly	BNBAI	● Informational	Aknowledged
BNBAI.sol:61,539,540,550,551,705,710,716	Use Assembly to Check Zero Address	Ownable	● Informational	Aknowledged
BNBAI.sol:139,142	Too Many Digits	BNBAI	● Informational	Aknowledged



## Code Security – Unchecked Call Return Value

Title	Severity	Location	Status
Unchecked Call Return Value	● Medium	BNBAI.sol:738	Aknowledged

### Description

The return value of low level calls and external calls (transfer, transferFrom and approve) should be verified since low level calls may fail and these three external function calls may only return false but not cause execution reverted once fail. If not properly handled, it might incur asset losses to users and the project party.

## Code Security – Use Safer Functions

Title	Severity	Location	Status
Use Safer Functions	● Low	BNBAI.sol:732,738	Aknowledged

### Description

When calling the transfer, transferFrom, and approve functions in the ERC20 contract, there are some contracts that are not fully implemented in accordance with the ERC20 standard. In order to more comprehensively judge whether the call result meets expectations or to be compatible with different ERC20 contracts, it is recommended to use the safeTransfer, safeTransferFrom, safeApprove function to call.

## Optimization Suggestion – Clarify Return Value

Title	Severity	Location	Status
Clarify Return Value	● Low	BNBAI.sol:478,508	Aknowledged

### Description

The returned variable is specified in the function signature, but it still calls the return statement to return a local variable defined in the function body or state variable. It is necessary to clarify whether the returned value meets expectations.



## Optimization Suggestion – Prefer .call() To send()/transfer()

Title	Severity	Location	Status
-------	----------	----------	--------

Prefer .call() To send()/transfer()

● Informational

BNBAI.sol:732

Aknowledged

### Description

The send or transfer function has a limit of 2300 gas.

## Optimization Suggestion – Recommend to Follow Code Layout Conventions

Title	Severity	Location	Status
-------	----------	----------	--------

Recommend to Follow Code Layout Conventions

● Informational

BNBAI.sol:5,38,115

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 – Parameters Should Be Declared as Calldata

Title	Severity	Location	Status
-------	----------	----------	--------

Parameters Should Be Declared as Calldata

● Informational

BNBAI.sol:698

Aknowledged

### Description

When the compiler parses the external or public function, it can directly read the function parameters from calldata. Setting it to other storage locations may waste gas. About 300-400 gas can be saved with optimization turned off while 120-150 gas can be saved vice versa.



## Optimization Suggestion – No Check of Address Params with Zero Address

Title	Severity	Location	Status
No Check of Address Params with Zero Address	<span>●</span> Informational	BNBAI.sol:202,331,340 ,353,357,736	Acknowledged

### 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	<span>●</span> Informational	BNBAI.sol:139,147	Acknowledged

### 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	<span>●</span> Informational	BNBAI.sol:624,649,654 ,659	Acknowledged

### 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 – 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	<span>●</span> Informational	BNBAI.sol:245,325,332,333,440,441,442,522,523,558,562,564,567,651,656,661,667,675,686	Acknowledged

### 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 ++i/--i Instead of i++/i--

Title	Severity	Location	Status
Use ++i/--i Instead of i++/i--	<span>●</span> Informational	BNBAI.sol:342,524,699	Acknowledged

### 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 – Use != 0 Instead of > 0 for Unsigned Integer Comparison

Title	Severity	Location	Status
Use != 0 Instead of > 0 for Unsigned Integer Comparison	<span>●</span> Informational	BNBAI.sol:552	Acknowledged

### 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	<span>●</span> Informational	BNBAI.sol:226,230,234,239,248,252,257,271,276,287,292,296,331,353,357,361,736	Acknowledged

### Description

Functions that are not called should be declared as external.

## Optimization Suggestion – Floating Pragma

Title	Severity	Location	Status
Floating Pragma	<span>●</span> Informational	BNBAI.sol:3	Acknowledged

### Description

Contracts should be deployed with fixed compiler version which has been tested thoroughly or make sure to lock the contract compiler version in the project configuration. Locked compiler version ensures that contracts will not be compiled by untested compiler version.

## Optimization Suggestion – Optimizable Return Statement

Title	Severity	Location	Status
Optimizable Return Statement	<span>●</span> Informational	BNBAI.sol:426,437,456,493,513	Acknowledged

### Description

The returned variable is specified in the function signature, but the return statement is still displayed in the function body, which will increase gas consumption.





## Optimization Suggestion – Use CustomError Instead of String

Title	Severity	Location	Status
Use CustomError Instead of String	<span>●</span> Informational	BNBAI.sol:52,61,265,281,301,312,319,320,325,332,341,539,540,550,551,552,553,559,705,710,716,721,731,737	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 – ReentrancyGuard Should Modify External Function

Title	Severity	Location	Status
ReentrancyGuard Should Modify External Function	<span>●</span> Informational	BNBAI.sol:623	Aknowledged

### Description

The reentrancy guard modifier should modify the external function, because reentrancy vulnerabilities often occur in external calls.

## Optimization Suggestion – Long String in revert/require

Title	Severity	Location	Status
Long String in revert/require	<span>●</span> Informational	BNBAI.sol:61,265,281,319,320,325,539,540,550,551,552,553,705,710,716,721,737	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	<span>●</span> Informational	BNBAI.sol:133,134	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	<span>●</span> Informational	BNBAI.sol:636,640,731	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	<span>●</span> Informational	BNBAI.sol:61,539,540,550,551,705,710,716	Aknowledged
------------------------------------	------------------------------	--	-------------

### Description

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



## Optimization Suggestion – Too Many Digits

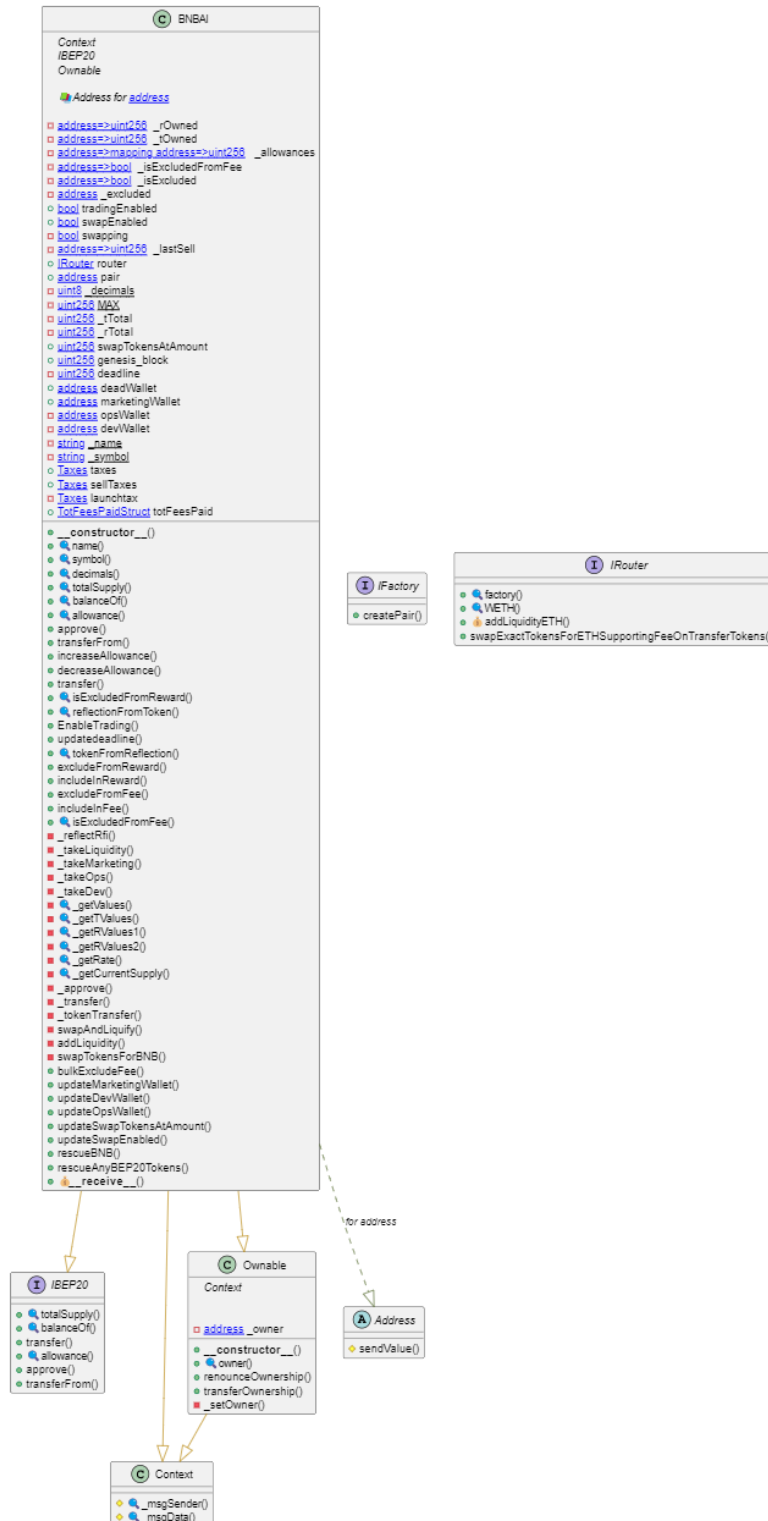
Title	Severity	Location	Status
Too Many Digits	 Informational	BNBAI.sol:139,142	Acknowledged

### Description

The number is too long, and it is easy to make mistakes when modifying and maintaining.



# PlantUML





# Appendix

## Finding Categories

### Security and Best Practices

1. **Unchecked Call Return Value:** Ensure that call return values are checked to prevent unexpected behavior in contract execution.
2. **Use Safer Functions:** Implement functions with known secure designs to reduce the risk of security vulnerabilities. Review functions for enhanced security measures.
3. **Clarify Return Value:** Clearly document and verify the return values of functions for better code understanding and reliability.
4. **Prefer .call() To send()/transfer():** Choose .call() over send()/transfer() for external contract calls to minimize security risks and ensure better interoperability.
5. **Recommend to Follow Code Layout Conventions:** Strictly adhere to established code layout conventions for improved code readability and maintainability.
6. **Parameters Should Be Declared as Calldata:** Declare parameters as calldata when possible for efficient function execution and reduced gas costs.
7. **No Check of Address Params with Zero Address:** Include checks in address parameters to ensure they are not the zero address, preventing potential vulnerabilities.
8. **Variables Should Be Constants:** Declare variables as constants when their values should not be modified after initialization.
9. **Use Shift Operation Instead of Mul/Div:** Employ shift operations over multiplication/division for enhanced gas efficiency and optimized contract execution.
10. **Cache State Variables that are Read Multiple Times within A Function:** Optimize gas usage by caching state variables read multiple times within a function.
11. **Use ++i/--i Instead of i++/i--:** Prefer pre-increment/pre-decrement (++i/--i) over post-increment/post-decrement (i++/i--) for potential gas savings.
12. **Use != 0 Instead of > 0 for Unsigned Integer Comparison:** Utilize != 0 for unsigned integer comparison instead of > 0 for better clarity and consistency.
13. **Function Visibility Can Be External:** Set function visibility to external if functions are only accessible from within the contract, enhancing gas efficiency.
14. **Floating Pragma:** Maintain a consistent Solidity pragma version for added contract security and compatibility.
15. **Optimizable Return Statement:** Optimize return statements for improved gas efficiency and contract performance.
16. **Use CustomError Instead of String:** Employ custom error codes instead of string error messages for more efficient contract operation.
17. **ReentrancyGuard Should Modify External Function:** Ensure that ReentrancyGuard modifications are applied to external functions for effective reentrancy protection.
18. **Long String in revert/require:** Avoid long strings in revert/require statements to optimize gas usage and contract efficiency.
19. **Variables Can Be Declared as Immutable:** Declare variables as immutable when their values remain constant after initialization for improved security and readability.
20. **Get Contract Balance of ETH in Assembly:** Utilize assembly to efficiently retrieve the contract's ETH balance.
21. **Use Assembly to Check Zero Address:** Implement optimized assembly checks to verify zero addresses efficiently.
22. **Too Many Digits:** Minimize the use of excessive digits for gas efficiency and readability in numeric values.



## 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://bnbai.xyz/>



## Network Security

High | 0 Attentions

## Application Security

High | 3 Attentions










## DNS Security

High | 3 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



8 Passed



3 Attention

**Missing X-Frame-Options Header**

YES

**Missing HSTS header**

NO

**Missing X-Content-Type-Options Header**

YES

**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

✓ 7 Passed

i 3 Attention

Missing SPF Record	NO	✓
Missing DMARC Record	YES	i
Missing DKIM Record	NO	✓
Ineffective SPF Record	YES	i
SPF Record Contains a Softfail Without DMARC	YES	i
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

8 Failed

X (Twitter)



PASS

Facebook

FAIL

Instagram

FAIL

TikTok

FAIL

YouTube

FAIL

Twich

FAIL

Telegram



PASS

Discord

FAIL

Medium

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

**NOT LAUNCHED**

Token Market Cap (estimate)

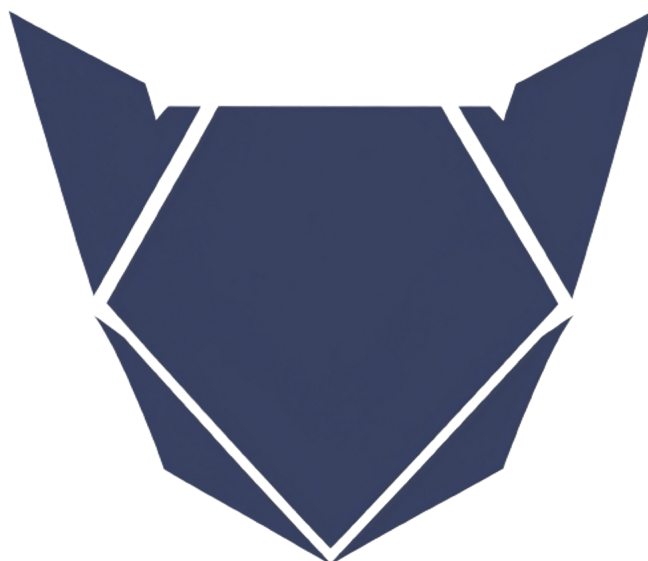
**\$13,762**

Token/Project Age

**0 Days**

## 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.





# Coin Tracker Analytics

## Status



CoinMarketCap

NO 



CoinGecko

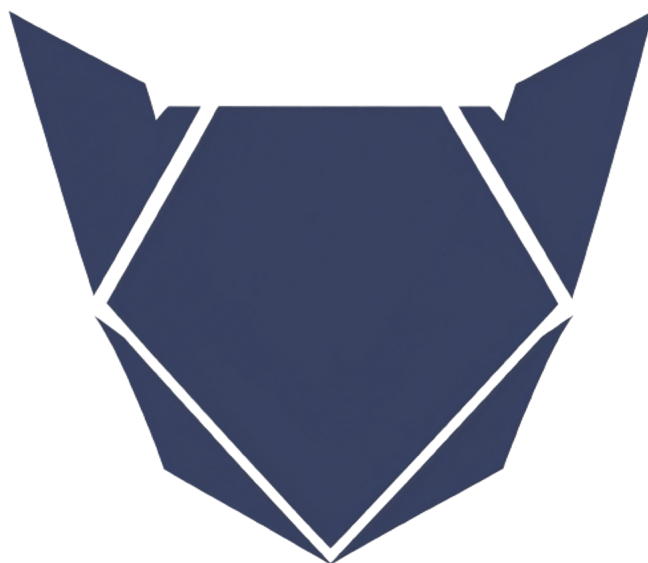
NO 

Others

NO 

## Recommendation

We highly recommend that the project consider integrating with multiple coin tracking platforms to expand its visibility within the cryptocurrency ecosystem. In particular, joining prominent platforms such as CoinMarketCap and CoinGecko can significantly benefit the project by increasing its reach and credibility.





# CEX Holding Analytics

## Status

Not available on any centralized cryptocurrency exchanges (CEX).

## Recommendation

To increase your project's visibility and liquidity, we recommend pursuing listings on centralized cryptocurrency exchanges. Here's a recommendation you can use:

We strongly advise the project team to actively pursue listings on reputable centralized cryptocurrency exchanges. Being listed on these platforms can offer numerous advantages, such as increased liquidity, exposure to a broader range of traders, and enhanced credibility within the crypto community.

To facilitate this process, we recommend the following steps:

1. **Research and Identify Suitable Exchanges:** Conduct thorough research to identify centralized exchanges that align with your project's goals and target audience. Consider factors such as trading volume, reputation, geographical reach, and compliance with regulatory requirements.
2. **Meet Compliance Requirements:** Ensure that your project is compliant with all necessary legal and regulatory requirements for listing on these exchanges. This may include Know Your Customer (KYC) verification, security audits, and legal documentation.
3. **Prepare a Comprehensive Listing Proposal:** Create a detailed and persuasive listing proposal for each exchange you intend to approach. This proposal should highlight the unique features and benefits of your project, as well as your commitment to compliance and security.
4. **Engage in Communication:** Establish open lines of communication with the exchange's listing team. Be prepared to address their questions, provide requested documentation, and work closely with their team to facilitate the listing process.
5. **Marketing and Community Engagement:** Promote your project within the exchange's community and among your own supporters to increase visibility and trading activity upon listing.
6. **Maintain Transparency:** Maintain transparency and provide regular updates to your community and potential investors about the progress of listing efforts.
7. **Be Patient and Persistent:** Listing processes on centralized exchanges can sometimes be lengthy. Be patient and persistent in your efforts, and consider seeking the assistance of experts or advisors with experience in exchange listings if necessary.
- 8.

Remember that listing on centralized exchanges can significantly impact your project's growth and market accessibility. By following these steps and maintaining a professional, compliant, and communicative approach, you can increase your chances of successfully getting listed on centralized exchanges.



# 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.

