SPHINXSHIELD

Security Assessment

Dragon Infinity Token

Nov 12th, 2023

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	15/19

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.

Audit Passed

PASSED

[illegible]

Table of Contents

Summary

This audit report is tailored for **Dragon Infinity Token**, aiming to uncover potential issues and vulnerabilities within the **Dragon Infinity Token** 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

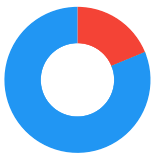
Project Summary

Project Name	Dragon Infinity Token
Blockchain	Binance Smart Chain
Language	Solidity
Codebase	https://bscscan.com/address/0xd6a92969a55661da1C17c2FcB13bA2aC6D60B3F4
Commit	04711125410a2aea95ae6f0f983c6eacb1b5f2a5a2e7b4f4bbcf2985165630cd

Audit Summary

Delivery Date	Nov 12th, 2023
Audit Methodology	Static Analysis, Manual Review
Key Components	DragonInfinity.sol

Vulnerability Summary



16 Total Findings

0 Resolved

0 Mitigated

0 Partially Resolved

16 Aknowledged

0 Declined

Vulnerability Level	Total	Pending	Declined	Aknowledged	Resolved
● High	3	0	0	3	0
● Medium	0	0	0	0	0
● Low	0	0	0	0	0
● Informational	13	0	0	13	0
● Discussion	0	0	0	0	0

Audit Scope

ID	File	KECCAK256 or SHA256 Checksum
DGT	DragonInfinity.sol	0x753232ebf9ad6c726878d3db5930f66c0c6b19aa3c143da32a04414187fe12d9

Understandings

Dragon Infinity Token is a Binance Smart Chain (BSC) token governed by the ERC20, ERC20Burnable, and Ownable contracts. It introduces unique features and functionalities to enhance user experience and security. Below is an overview of its key components and operations:

Token Information

- Token Name: Dragon Infinity Token
- Symbol: DRIN
- Decimals: 18
- Total Supply: 1,000,000,000 DRIN

Tax Distribution

Dragon Infinity Token employs a dynamic tax distribution system on transactions, with fees divided into various components:

1. Liquidity Fee: Collected for providing liquidity, its value is set by the owner.
2. Team Fee: A portion allocated to the project's team, with an adjustable fee by the owner.
3. Marketing Fee: Allocated for marketing efforts, configurable by the owner.
4. Dev Fee: Reserved for development purposes, its value can be set by the owner.
5. Burn Fee: Tokens are burned, reducing the total supply, with an adjustable fee.
6. Total Fee: The sum of all fees, determining the overall fee distribution.
7. Fee Denominator: The denominator used in fee calculations, typically set to 100..

Fee Management

The contract provides flexibility in managing fees:

- Set Tax: The owner can configure liquidity, team, marketing, dev, and burn fees, along with the fee denominator.
- Set Fee Multipliers: Adjust the percentage of fees for buy, sell, and transfer transactions.
- Set Fee Receivers: Designate addresses receiving fee components (auto-liquidity, marketing, dev, and team fees).

Tax Exemption

Owners have the authority to exempt specific addresses from fees, enabling fee waivers for whitelisted wallets or contracts.

Ownership and Authorization

The contract owner can authorize specific addresses with privileged functions. These functions are restricted by the onlyOwner modifier and are crucial for configuring the contract and address attributes.

Transaction Limits

To prevent excessive token movement, the contract enforces transaction limits, ensuring users adhere to defined limits.

Swap Mechanism

Dragon Infinity Token utilizes a swap mechanism to manage liquidity. When a token threshold is reached, a portion of the balance is swapped to BB tokens via the PancakeSwap Router. The remaining balance is then supplied to the DRIN-BNB liquidity pool.

Open Trading

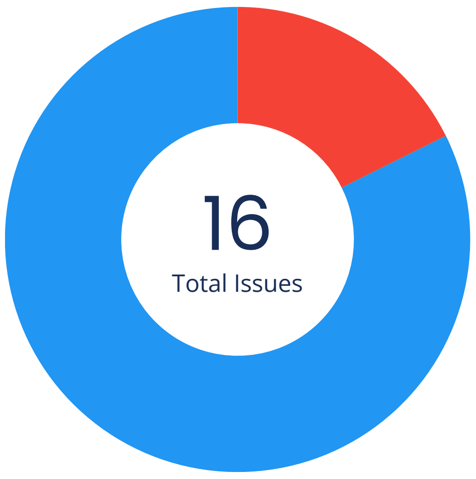
Trading can be restricted based on conditions defined by the owner, ensuring that trading remains closed until specific requirements are met.

Additional Functionality

The contract includes various functions, such as clearing stuck ETH, clearing tokens, and more, contributing to the overall efficiency and robustness of the Dragon Infinity Token project.

This overview provides insights into the core features and functions of the Dragon Infinity Token contract on the Binance Smart Chain. Please note that the contract plays a vital role in governing fee structures, liquidity, and user interactions within the Dragon Infinity Token project.

Findings



● High - 3
● Medium - 0
● Low - 0
● Informational - 13
● Discussion - 0

Location	Title	Scope	Severity	Status
DragonInfinity.sol:906	Integer Overflow/Underflow	ERC20	● High	Aknowledged
DragonInfinity.sol:851	Integer Overflow/Underflow	ERC20	● High	Aknowledged
DragonInfinity.sol:959	Integer Overflow/Underflow	ERC20	● High	Aknowledged
DragonInfinity.sol:1175	Comparison With Literal Boolean	DragonInfinity	● Informational	Aknowledged
DragonInfinity.sol:3,499,1090	Recommend to Follow Code Layout Conventions	Ownable	● Informational	Aknowledged
DragonInfinity.sol:1169,1209,1217,1225,1234	No Check of Address Params with Zero Address	DragonInfinity	● Informational	Aknowledged
DragonInfinity.sol:1102	Variables Should Be Constants	DragonInfinity	● Informational	Aknowledged

Location	Title	Scope	Severity	Status
DragonInfinity.sol:1090	No Need To Use SafeMath in Solidity Contract of Version 0.8.0 and Above	DragonInfinity	● Informational	Aknowledged
DragonInfinity.sol:734,742,759,773,785,808,830,849,869	Function Visibility Can Be External	ERC20	● Informational	Aknowledged
DragonInfinity.sol:3,203,240,457,484,568,648,677,1043,1081	Floating Pragma	Global	● Informational	Aknowledged
DragonInfinity.sol:3	Usage of Outdated Compiler	Global	● Informational	Aknowledged
DragonInfinity.sol:530,549,872,895,896,901,924,950,955,981,982,999,1175,1199,1200	Use CustomError Instead of String	Ownable	● Informational	Aknowledged
DragonInfinity.sol:1102	Unused State Variables	DragonInfinity	● Informational	Aknowledged
DragonInfinity.sol:549,872,895,896,901,950,955,981,982	Long String in revert/require	Ownable	● Informational	Aknowledged
DragonInfinity.sol:1020,1036	Empty Function Body	ERC20	● Informational	Aknowledged
DragonInfinity.sol:549,895,896,924,950,981,982	Use Assembly to Check Zero Address	Ownable	● Informational	Aknowledged

Code Security - Integer Overflow/Underflow

Title	Severity	Location	Status
Integer Overflow/Underflow	🔴 High	DragonInfinity.sol:906	Aknowledged

Description

An overflow/underflow may happen when an arithmetic operation reaches the maximum or minimum size of a type.

Code Security - Integer Overflow/Underflow

Title	Severity	Location	Status
Integer Overflow/Underflow	🔴 High	DragonInfinity.sol:851	Aknowledged

Description

An overflow/underflow may happen when an arithmetic operation reaches the maximum or minimum size of a type.

Code Security - Integer Overflow/Underflow

Title	Severity	Location	Status
Integer Overflow/Underflow	🔴 High	DragonInfinity.sol:959	Aknowledged

Description

An overflow/underflow may happen when an arithmetic operation reaches the maximum or minimum size of a type.

Optimization Suggestion - Comparison With Literal Boolean

Title	Severity	Location	Status
Comparison With Literal Boolean	● Informational	DragonInfinity.sol:1175	Aknowledged

Description

Boolean constants can be used directly and do not need to be compare to true or false.

Optimization Suggestion - Recommend to Follow Code Layout Conventions

Title	Severity	Location	Status
Recommend to Follow Code Layout Conventions	● Informational	DragonInfinity.sol:3,499,1090	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	DragonInfinity.sol:1169,1209,1217,1225,1234	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	DragonInfinity.sol:1102	Aknowledged

Description

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

Optimization Suggestion - No Need To Use SafeMath in Solidity Contract of Version 0.8.0 and Above

Title	Severity	Location	Status
No Need To Use SafeMath in Solidity Contract of Version 0.8.0 and Above	● Informational	DragonInfinity.sol:1090	Aknowledged

Description

In solidity 0.8.0 and above, the compiler has its own overflow checking function, so there is no need to use the SafeMath library to prevent overflow.

Optimization Suggestion - Function Visibility Can Be External

Title	Severity	Location	Status
Function Visibility Can Be External	● Informational	DragonInfinity.sol:734, 742,759,773,785,808,830,849,869	Aknowledged

Description

Functions that are not called should be declared as external.

Optimization Suggestion - Floating Pragma

Title	Severity	Location	Status
Floating Pragma	● Informational	DragonInfinity.sol:3,203,240,457,484,568,648,677,1043,1081	Aknowledged

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 - Usage of Outdated Compiler

Title	Severity	Location	Status
Usage of Outdated Compiler	● Informational	DragonInfinity.sol:3	Aknowledged

Description

The outdated compiler version (below solidity 0.8.0) is used in the contract, and there may be some security vulnerabilities.

Optimization Suggestion - Use CustomError Instead of String

Title	Severity	Location	Status
Use CustomError Instead of String	● Informational	DragonInfinity.sol:530,549,872,895,896,901,924,950,955,981,982,999,1175,1199,1200	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 - Unused State Variables

Title	Severity	Location	Status
Unused State Variables	● Informational	DragonInfinity.sol:1102	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	DragonInfinity.sol:549,872,895,896,901,950,955,981,982	Aknowledged

Description

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

Optimization Suggestion - Empty Function Body

Title	Severity	Location	Status
Empty Function Body	● Informational	DragonInfinity.sol:1020,1036	Aknowledged

Description

The body of this function is empty.

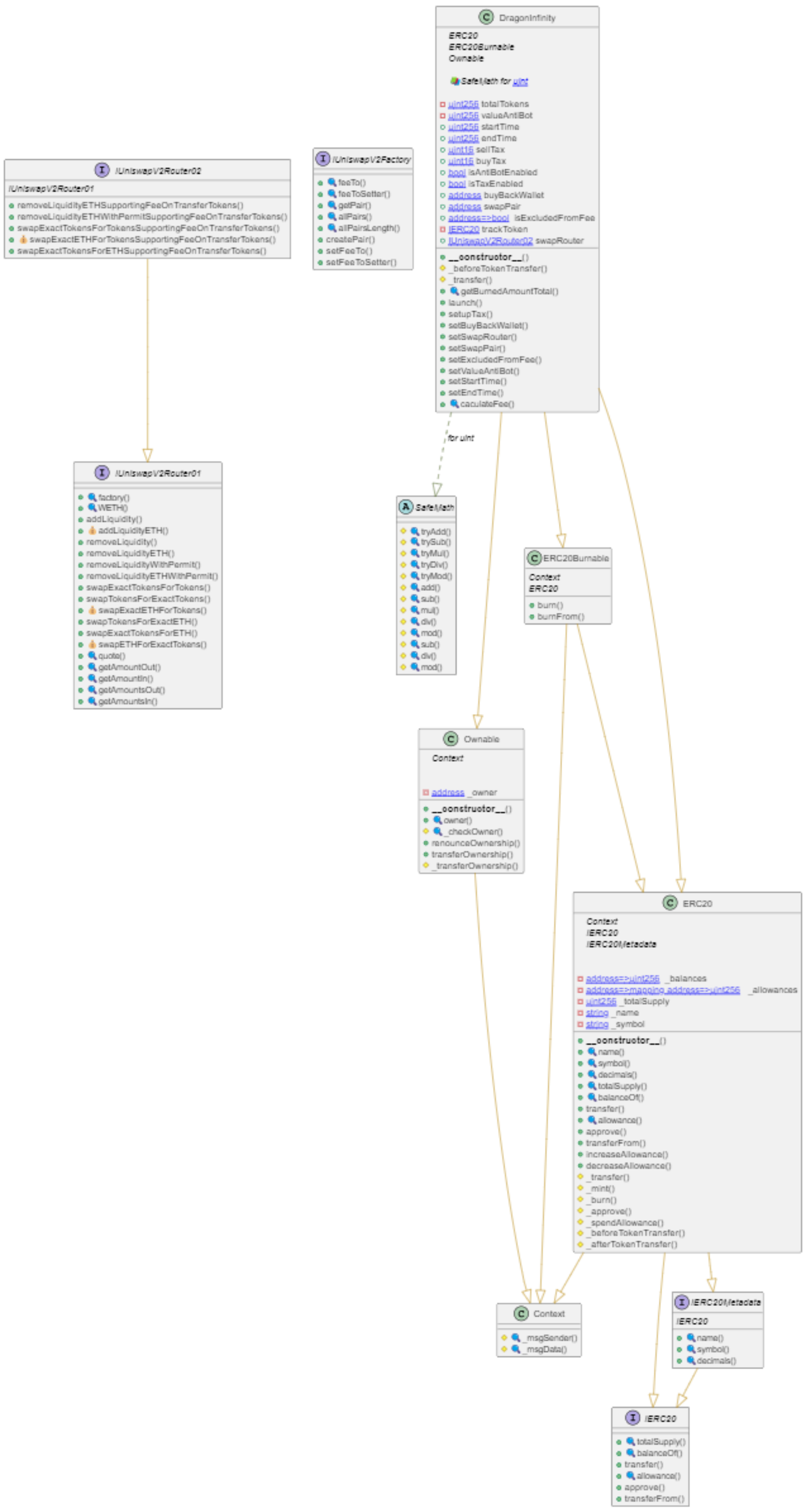
Optimization Suggestion - Use Assembly to Check Zero Address

Title	Severity	Location	Status
Use Assembly to Check Zero Address	🔵 Informational	DragonInfinity.sol:549,895,896,924,950,981,982	Aknowledged

Description

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

PlantUML


DragonInfinity
ERC20
ERC20Burnable
Ownable
SafeMath for uint
uint256 totalTokens
uint256 valueAntiBot
uint256 startTime
uint256 endTime
uint16 sellTax
uint16 buyTax
bool isAntiBotEnabled
bool isTaxEnabled
address buyBackWallet
address swapPair
address=>bool isExcludedFromFee
IERC20 trackToken
IUniswapV2Router02 swapRouter
__constructor__()
_beforeTokenTransfer()
_transfer()
getBurnedAmountTotal()
launch()
setupTax()
setBuyBackWallet()
setSwapRouter()
setSwapPair()
setExcludedFromFee()
setValueAntiBot()
setStartTime()
setEndTime()
caculateFee()
for uint
IUniswapV2Router02
IUniswapV2Router01
removeLiquidityETHSupportingFeeOnTransferTokens()
removeLiquidityETHWithPermitSupportingFeeOnTransferTokens()
swapExactTokensForTokensSupportingFeeOnTransferTokens()
swapExactETHForTokensSupportingFeeOnTransferTokens()
swapExactTokensForETHSupportingFeeOnTransferTokens()
IUniswapV2Factory
feeTo()
feeToSetter()
getPair()
allPairs()
allPairsLength()
createPair()
setFeeTo()
setFeeToSetter()
IUniswapV2Router01
factory()
WETH()
addLiquidity()
addLiquidityETH()
removeLiquidity()
removeLiquidityETH()
removeLiquidityWithPermit()
removeLiquidityETHWithPermit()
swapExactTokensForTokens()
swapTokensForExactTokens()
swapExactETHForTokens()
swapTokensForExactETH()
swapExactTokensForETH()
swapETHForExactTokens()
quote()
getAmountOut()
getAmountIn()
getAmountsOut()
getAmountsIn()
SafeMath
tryAdd()
trySub()
tryMul()
tryDiv()
tryMod()
add()
sub()
mul()
div()
mod()
sub()
div()
mod()
ERC20Burnable
Context
ERC20
burn()
burnFrom()
Ownable
Context
address _owner
__constructor__()
owner()
_checkOwner()
renounceOwnership()
transferOwnership()
_transferOwnership()
ERC20
Context
IERC20
IERC20Metadata
address=>uint256 _balances
address=>mapping address=>uint256 _allowances
uint256 _totalSupply
string _name
string _symbol
__constructor__()
name()
symbol()
decimals()
totalSupply()
balanceOf()
transfer()
allowance()
approve()
transferFrom()
increaseAllowance()
decreaseAllowance()
_transfer()
_mint()
_burn()
_approve()
_spendAllowance()
_beforeTokenTransfer()
_afterTokenTransfer()
Context
_msgSender()
_msgData()
IERC20Metadata
IERC20
name()
symbol()
decimals()
IERC20
totalSupply()
balanceOf()
transfer()
allowance()
approve()
transferFrom()

Appendix

Finding Categories

Security and Best Practices

1. Integer Overflow/Underflow: Smart contracts should implement safeguards against integer overflow/underflow to prevent unintended behavior and vulnerabilities.
2. Comparison With Literal Boolean: Review code for comparisons with literal booleans, ensuring accurate logic and avoiding unnecessary complexity.
3. Recommend to Follow Code Layout Conventions: Strict adherence to established code layout conventions can significantly improve code readability and maintainability.
4. No Check of Address Params with Zero Address: Verification of address parameters should include checks to ensure that the address is not the zero address, preventing potential vulnerabilities.
5. Variables Should Be Constants: Declare variables as constants when their values should not be modified after initialization.
6. No Need To Use SafeMath in Solidity Contract of Version 0.8.0 and Above: Solidity versions 0.8.0 and above feature built-in overflow and underflow protection, minimizing the necessity of SafeMath library usage.
7. Function Visibility Can Be External: Enhance gas efficiency by setting functions to external visibility if they are accessible only from within the contract.
8. Floating Pragma: Maintain a consistent Solidity pragma version for added contract security and compatibility.
9. Usage of Outdated Compiler: Employ the latest Solidity compiler version to leverage security updates and improvements.
10. Use CustomError Instead of String: Opt for custom error codes instead of string error messages for more efficient contract operation.
11. Unused State Variables: Eliminate unused state variables to enhance code clarity and efficiency.
12. Long String in revert/require: Avoid long strings in revert/require statements to optimize gas usage and contract efficiency.
13. Empty Function Body: Functions should not contain empty bodies, as this can introduce vulnerabilities and reduce code readability.
14. Use Assembly to Check Zero Address: Implement optimized assembly checks to verify zero addresses 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://dragoninfinity.app/

Network Security

High | 0 Attentions

Application Security

High | 3 Attentions

DNS Security

High | 4 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

✓ 6 Passed   ⓘ 4 Attention

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

✓ 3 Passed   ⓘ 7 Failed

X (Twitter)	PASS ✓
Facebook	FAIL ✗
Instagram	FAIL ✗
TikTok	FAIL ✗
YouTube	FAIL ✗
Twich	FAIL ✗
Telegram	PASS ✓
Discord	FAIL ✗
Medium	PASS ✓
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)	NOT AVAILABLE
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.