SPHINXSHIELD

Security Assessment

Gravitas

Nov 15th, 2023

Evaluation Outcomes

Security Score

Review	Score
Overall Score	88/100
Auditor Score	82/100

Review by Section	Score
Manual Scan Score	45/57
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 **Gravitas**, aiming to uncover potential issues and vulnerabilities within the **Gravitas** 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

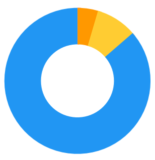
Project Summary

Project Name	Gravitas
Blockchain	Binance Smart Chain
Language	Solidity
Codebase	https://bscscan.com/address/0xff2e1a8334bd43eee3fb61f6b524fc004a59fe5a
Commit	36b6b05bbed7cc62597d3079fff1455c8638938b1383ba0410bbdbc600de42bb

Audit Summary

Delivery Date	Nov 15th, 2023
Audit Methodology	Static Analysis, Manual Review
Key Components	Gravitas.sol

Vulnerability Summary



22
Total Findings

0	0	0	22	0
Resolved	Mitigated	Partially Resolved	Aknowledged	Declined

Vulnerability Level	Total	Pending	Declined	Aknowledged	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
GVT	Gravitas.sol	0x042c9b607382592a31b70e6c28eb3b4f5c192a5b52a7f8060d35d3513d7f5a92

Understandings

Gravitas is a decentralized finance (DeFi) token deployed on the Binance Smart Chain (BSC). The Gravitas contract incorporates various functions and features to manage its operations, including tax distribution, liquidity acquisition, and privileged functions for modifying contract parameters. Here's a breakdown of key components and functionalities within the Gravitas contract:

Token Information

- Token Name: Gravitas
- Symbol: Gravitas
- Decimals: 9
- Total Supply: 1,000,000,000 Gravitas

Tax Distribution

Gravitas transactions incur a total fee, which is distributed across various components:

- RFI Fee: A portion of the fee is redistributed to holders. The RFI fee is not applicable during the launch phase.
- Marketing Fee: Part of the fee is allocated to marketing efforts. The marketing fee is adjustable by the owner.
- Operations Fee: A portion of the fee goes to the project's operations. The operations fee is adjustable by the owner.
- Liquidity Fee: This fee is collected for providing liquidity to the token. The liquidity fee is adjustable by the owner.
- Development Fee: A portion of the fee is allocated for development purposes. The development fee is adjustable by the owner.
- Total Fee: The sum of the above-mentioned fees forms the total fee collected in each transaction. It is used to calculate the overall fee distribution.
- Fee Denominator: The denominator used in fee calculations, usually set to 200.

Fee Management

The contract allows the owner to manage various fees:

- Set Tax: The owner can configure the marketing fee, operations fee, liquidity fee, development fee, and RFI fee, as well as the fee denominator. These adjustments enable flexibility in managing the fee structure.
- Set Fee Multipliers: The owner can set multipliers to adjust the percentage of fees for buy, sell, and transfer transactions.
- Set Fee Receivers: The addresses that receive the various fee components (marketing, operations, liquidity, development, and RFI fees) can be set by the owner.

Tax Exemption

The contract allows the owner to exempt specific addresses from fees, providing a mechanism for fee exemption to certain addresses. This can be used for whitelisting specific wallets or contracts.

Ownership and Authorization

The contract owner can 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, ensuring that users do not make transactions that exceed the defined limits.

Swap Mechanism

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

Open Trading

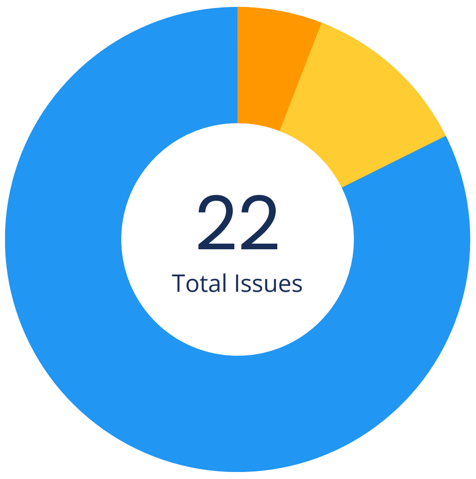
Trading can be restricted based on conditions defined by the owner. This ensures that trading remains closed until specific requirements are met.

Additional Functionality

The contract includes other functions, such as rescuing BNB sent to the contract by mistake and rescuing any BEP-20 tokens sent to the contract by mistake.

This understanding provides insights into the key features and functions of the Gravitas contract deployed on the Binance Smart Chain. Please note that the contract is a vital component of the Gravitas project's infrastructure, governing various aspects of its operation, including fees, liquidity, and user interactions.

Findings



● High - 0
● Medium - 1
● Low - 2
● Informational - 19
● Discussion - 0

Location	Title	Scope	Severity	Status
Gravitas.sol:738	Unchecked Call Return Value	Gravitas	● Medium	Aknowledged
Gravitas.sol:732,738	Use Safer Functions	Gravitas	● Low	Aknowledged
Gravitas.sol:478,508	Clarify Return Value	Gravitas	● Low	Aknowledged
Gravitas.sol:732	Prefer .call() To send()/transfer()	Gravitas	● Informational	Aknowledged
Gravitas.sol:5,38,115	Recommend to Follow Code Layout Conventions	IBEP20	● Informational	Aknowledged
Gravitas.sol:698	Parameters Should Be Declared as Calldata	Gravitas	● Informational	Aknowledged
Gravitas.sol:202,331,340,353,357,736	No Check of Address Params with Zero Address	Gravitas	● Informational	Aknowledged

Location	Title	Scope	Severity	Status
Gravitas.sol:139,147	Variables Should Be Constants	Gravitas	● Informational	Aknowledged
Gravitas.sol:624,649,654,659	Use Shift Operation Instead of Mul/Div	Gravitas	● Informational	Aknowledged
Gravitas.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	Gravitas	● Informational	Aknowledged
Gravitas.sol:342,524,699	Use ++i/--i Instead of i++/i--	Gravitas	● Informational	Aknowledged
Gravitas.sol:552	Use != 0 Instead of > 0 for Unsigned Integer Comparison	Gravitas	● Informational	Aknowledged
Gravitas.sol:226,230,234,239,248,252,257,271,276,287,292,296,331,353,357,361,736	Function Visibility Can Be External	Gravitas	● Informational	Aknowledged
Gravitas.sol:3	Floating Pragma	Global	● Informational	Aknowledged
Gravitas.sol:426,437,456,493,513	Optimizable Return Statement	Gravitas	● Informational	Aknowledged
Gravitas.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	Aknowledged

Location	Title	Scope	Severity	Status
Gravitas.sol:623	ReentrancyGuard Should Modify External Function	Gravitas	🔵 Informational	Aknowledged
Gravitas.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	Aknowledged
Gravitas.sol:133,134	Variables Can Be Declared as Immutable	Gravitas	🔵 Informational	Aknowledged
Gravitas.sol:636,640,731	Get Contract Balance of ETH in Assembly	Gravitas	🔵 Informational	Aknowledged
Gravitas.sol:61,539,540,550,551,705,710,716	Use Assembly to Check Zero Address	Ownable	🔵 Informational	Aknowledged
Gravitas.sol:139,142	Too Many Digits	Gravitas	🔵 Informational	Aknowledged

Code Security - Unchecked Call Return Value

Title	Severity	Location	Status
Unchecked Call Return Value	🟠 Medium	Gravitas.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	Gravitas.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	Gravitas.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	Gravitas.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	Gravitas.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	Gravitas.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	● Informational	Gravitas.sol:202,331,340,353,357,736	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	Gravitas.sol:139,147	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	Gravitas.sol:624,649,654,659	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 - 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	Gravitas.sol:245,325,332,333,440,441,442,522,523,558,562,564,567,651,656,661,667,675,686	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 ++i/--i Instead of i++/i--

Title	Severity	Location	Status
Use ++i/--i Instead of i++/i--	🔵 Informational	Gravitas.sol:342,524,699	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 - Use != 0 Instead of > 0 for Unsigned Integer Comparison

Title	Severity	Location	Status
Use != 0 Instead of > 0 for Unsigned Integer Comparison	🔵 Informational	Gravitas.sol:552	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	Gravitas.sol:226,230,234,239,248,252,257,271,276,287,292,296,331,353,357,361,736	Aknowledged

Description

Functions that are not called should be declared as external.

Optimization Suggestion - Floating Pragma

Title	Severity	Location	Status
Floating Pragma	🔵 Informational	Gravitas.sol:3	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 - Optimizable Return Statement

Title	Severity	Location	Status
Optimizable Return Statement	🔵 Informational	Gravitas.sol:426,437,456,493,513	Aknowledged

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	● Informational	Gravitas.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	● Informational	Gravitas.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	● Informational	Gravitas.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	● Informational	Gravitas.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	● Informational	Gravitas.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	● Informational	Gravitas.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.

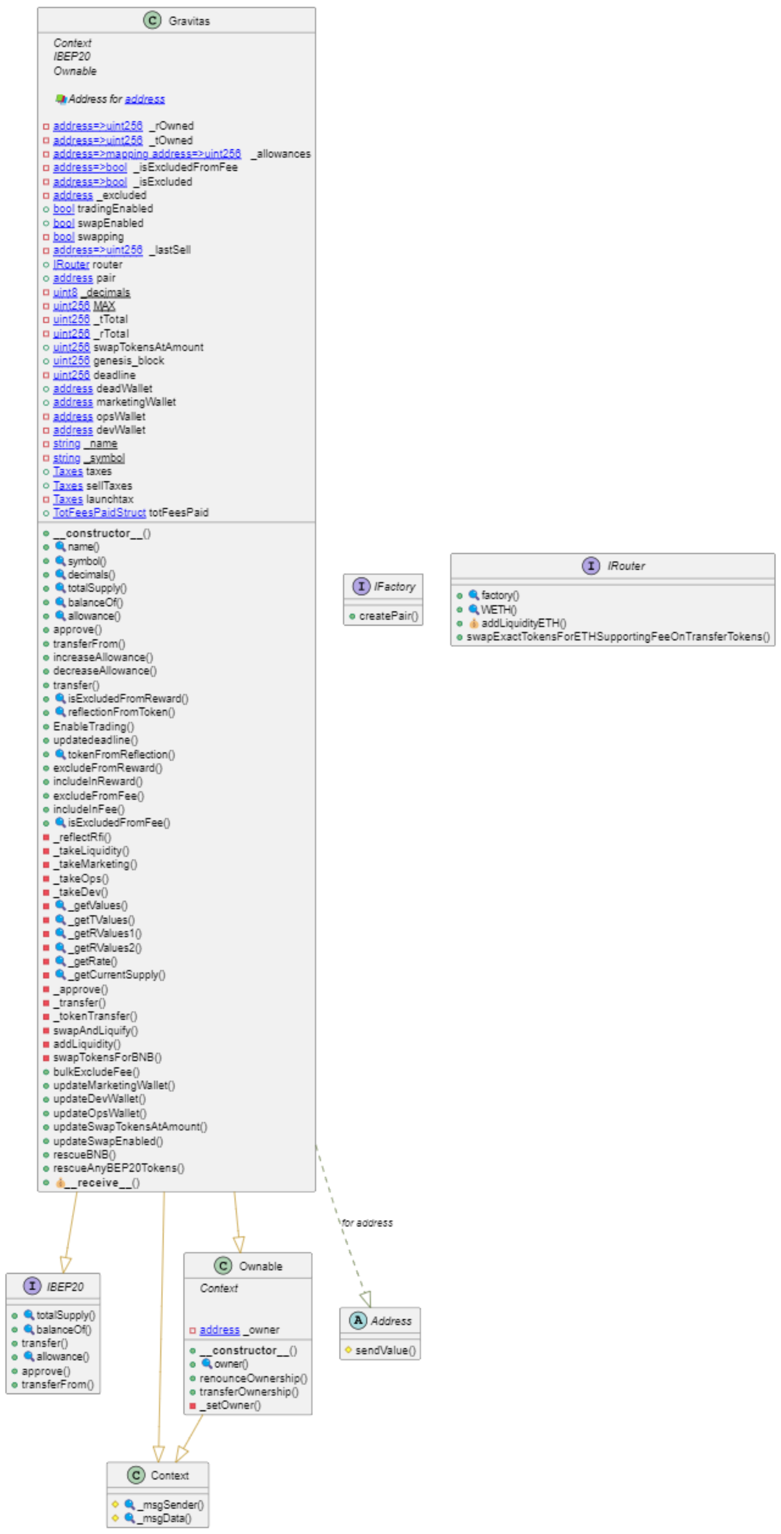
Optimization Suggestion - Too Many Digits

Title	Severity	Location	Status
Too Many Digits	🔵 Informational	Gravitas.sol:139,142	Aknowledged

Description

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

PlantUML


Gravitas
Context
IBEP20
Ownable
Address for address
address=>uint256 _rOwned
address=>uint256 _tOwned
address=>mapping address=>uint256 _allowances
address=>bool _isExcludedFromFee
address=>bool _isExcluded
address _excluded
bool tradingEnabled
bool swapEnabled
bool swapping
address=>uint256 _lastSell
IRouter router
address pair
uint8 _decimals
uint256 MAX
uint256 _tTotal
uint256 _rTotal
uint256 swapTokensAtAmount
uint256 genesis_block
uint256 deadline
address deadWallet
address marketingWallet
address opsWallet
address devWallet
string _name
string _symbol
Taxes taxes
Taxes sellTaxes
Taxes launchtax
TotFeesPaidStruct totFeesPaid
__constructor__()
name()
symbol()
decimals()
totalSupply()
balanceOf()
allowance()
approve()
transferFrom()
increaseAllowance()
decreaseAllowance()
transfer()
isExcludedFromReward()
reflectionFromToken()
EnableTrading()
updatedeadline()
tokenFromReflection()
excludeFromReward()
includeInReward()
excludeFromFee()
includeInFee()
isExcludedFromFee()
_reflectRfi()
_takeLiquidity()
_takeMarketing()
_takeOps()
_takeDev()
_getValues()
_getTValues()
_getRValues1()
_getRValues2()
_getRate()
_getCurrentSupply()
_approve()
_transfer()
_tokenTransfer()
swapAndLiquify()
addLiquidity()
swapTokensForBNB()
bulkExcludeFee()
updateMarketingWallet()
updateDevWallet()
updateOpsWallet()
updateSwapTokensAtAmount()
updateSwapEnabled()
rescueBNB()
rescueAnyBEP20Tokens()
__receive__()
IFactory
createPair()
IRouter
factory()
WETH()
addLiquidityETH()
swapExactTokensForETHSupportingFeeOnTransferTokens()
IBEP20
totalSupply()
balanceOf()
transfer()
allowance()
approve()
transferFrom()
Ownable
Context
address _owner
__constructor__()
owner()
renounceOwnership()
transferOwnership()
_setOwner()
for address
Address
sendValue()
Context
_msgSender()
_msgData()

Appendix

Finding Categories

Security and Best Practices

1. Unchecked Call Return Value: Exercise caution when handling unchecked call return values to prevent unexpected behavior. Verify and validate return values before proceeding.
2. Use Safer Functions: Prioritize the use of functions known for their secure design to mitigate potential security vulnerabilities. Review functions thoroughly for enhanced security.
3. Clarify Return Value: Clearly document and define the expected return values for functions to avoid confusion and potential misuse.
4. Prefer .call() To send()/transfer(): Opt for the .call() method over send() or transfer() when making external contract calls to minimize security risks.
5. Recommend to Follow Code Layout Conventions: Adhere to established code layout conventions to significantly improve code readability and maintainability.
6. Parameters Should Be Declared as Calldata: Declare parameters as calldata to enhance security and prevent unintended modifications.
7. No Check of Address Params with Zero Address: Implement checks to verify that address parameters are not the zero address to enhance security.
8. Variables Should Be Constants: Declare variables as constants when their values remain constant throughout the contract execution.
9. Use Shift Operation Instead of Mul/Div: Leverage shift operations instead of multiplication and division for improved efficiency and reduced gas costs.
10. Cache State Variables that are Read Multiple Times within a Function: Optimize gas usage by caching state variables that are read multiple times within a function.
11. Use ++i/--i Instead of i++/i--: Employ the ++i/--i increment and decrement operators instead of i++/i-- for enhanced gas efficiency.
12. Use != 0 Instead of > 0 for Unsigned Integer Comparison: Use != 0 for unsigned integer comparisons instead of > 0 to ensure code clarity and consistency.
13. Function Visibility Can Be External: Enhance gas efficiency by setting functions to external visibility if they are accessible only from within the contract.
14. Floating Pragma: Maintain consistency in the Solidity pragma version for added contract security.
15. Optimizable Return Statement: Optimize return statements for improved gas efficiency without compromising code clarity.
16. Use CustomError Instead of String: Choose custom error codes over string error messages for more efficient contract operation.
17. ReentrancyGuard Should Modify External Function: Ensure that the ReentrancyGuard modifier is applied to functions modifying external contracts to prevent reentrancy attacks.
18. Long String in revert/require: Optimize long revert or require strings to reduce gas usage and enhance contract efficiency.
19. Variables Can Be Declared as Immutable: Declare variables as immutable if their values remain constant after initialization to enhance 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: Exercise caution with excessively large numbers to avoid potential overflow or precision issues.

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

⚠ NO WEBSITE AVAILABLE

Network Security

N/A | ⚠

Application Security

N/A | ⚠

DNS Security

N/A | ⚠

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	**2023.11.12 13:00 (UTC)**
Token Market Cap (estimate)	**$2.33K**
Token/Project Age	**3 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.