SPHINXSHIELD

Security Assessment

Sakai Vault

Feb 4th, 2024

Evaluation Outcomes

Security Score

Review	**Score**
Overall Score	93/100
Auditor Score	88/100

Review by Section	**Score**
Manual Scan Score	52/57
Advance Check Score	17/19

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 **Sakai Vault**, aiming to uncover potential issues and vulnerabilities within the **Sakai Vault** 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	Sakai Vault
Blockchain	Binance Smart Chain
Language	Solidity
Codebase	https://bscscan.com/token/0x43B35e89d15B91162Dea1C51133C4c93bdd1C4aF
Commit	2e52d6f392b63f1a818e6b3f150b87c285db8639d3d36e6e26ed981d46b72b37

Audit Summary

Delivery Date	Jan 11th, 2024
Audit Methodology	Static Analysis, Manual Review
Key Components	BabyDragon.sol

Vulnerability Summary

18 Total Findings

0	0	0	18	0
Resolved	Mitigated	Partially Resolved	Aknowledged	Declined

Vulnerability Level	Total	Pending	Declined	Aknowledged	Resolved
🔴 High	0	0	0	0	0
🟠 Medium	4	0	0	4	0
🟡 Low	1	0	0	1	0
🔵 Informational	13	0	0	13	0
🟢 Discussion	0	0	0	0	0

Audit Scope

ID	File	KECCAK256 or SHA256 Checksum
SKV	Sakai.sol	0x2a5455a652fa2aa1385b963a1472cf9dee74f6251c8482b392844ed5b25c163e

Understandings

Sakai Vault is a decentralized perpetual exchange and the next generation of DeFi. The Sakai Vault contract, deployed on the Binance Smart Chain (BSC) blockchain, includes various functions and mechanisms to facilitate its operations. Here's a breakdown of its key components and functionalities

Token Information

- Token Name: Sakai Vault
- Symbol: SAKAI
- Decimals: 18
- Total Supply: 8,000,000

Fee Management

- Swap Tax: A configurable tax percentage applied to transactions.
- Denominator: The denominator used in fee calculations, usually set to 10,000.

Ownership and Authorization

- Owner: The contract owner can authorize specific addresses to access privileged functions, restricted by the onlyOwner modifier. These functions are used for configuring the contract and address attributes.

Transaction Limits

The contract enforces transaction limits to prevent excessive token movement, ensuring adherence to predefined limits.

Swap Mechanism

Sakai Vault 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 USDT via the Uniswap Router. This mechanism helps maintain liquidity and stabilize the token price.

Open Trading

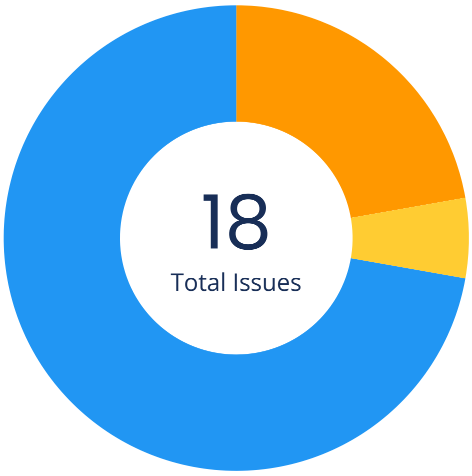
Trading can be restricted based on conditions defined by the owner, ensuring that trading remains closed until specific requirements are met.

Additional Functionality

- The contract includes functions for clearing stuck ETH, clearing tokens, and more, enhancing its operational flexibility and robustness.

This understanding provides insights into the key features and functions of the Sakai Vault contract, an essential component of the Sakai Vault project's infrastructure, governing various aspects of its operation.

Findings



● High - 0
● Medium - 4
● Low - 1
● Informational - 13
● Discussion - 0

Location	Title	Scope	Severity	Status
Sakai.sol:826	Unauthenticated Storage Access	ERC20	● Medium	Aknowledged
Sakai.sol:860	Unauthenticated Storage Access	ERC20	● Medium	Aknowledged
Sakai.sol:887	Unauthenticated Storage Access	ERC20	● Medium	Aknowledged
Sakai.sol:910	Unauthenticated Storage Access	ERC20	● Medium	Aknowledged
Sakai.sol:1461	Use Safer Functions	SAKAI	● Low	Aknowledged
Sakai.sol:1461	Prefer .call() To send()/transfer()	SAKAI	● Informational	Aknowledged
Sakai.sol:2,396,137 3	Recommend to Follow Code Layout Conventions	Ownable	● Informational	Aknowledged
Sakai.sol:1138	Unused Events	IUniswapV2F actory	● Informational	Aknowledged

Location	Title	Scope	Severity	Status
Sakai.sol:1534,1551	No Check of Address Params with Zero Address	SAKAI	🔵 Informational	Aknowledged
Sakai.sol:1628	Use Shift Operation Instead of Mul/Div	SAKAI	🔵 Informational	Aknowledged
Sakai.sol:1564	Redundant Getter Function	SAKAI	🔵 Informational	Aknowledged
Sakai.sol:910,934,1437,1547,1551,1564	Function Visibility Can Be External	ERC20	🔵 Informational	Aknowledged
Sakai.sol:430,449,687,941,971,972,977,1003,1064,1065,1086,1458,1470,1474,1484,1488,1498,1502,1503,1513,1517,1528,1538,1555,1660	Use CustomError Instead of String	Ownable	🔵 Informational	Aknowledged
Sakai.sol:1625	ReentrancyGuard Should Modify External Function	SAKAI	🔵 Informational	Aknowledged
Sakai.sol:1430,1471,1485,1499	Cache State Variables That Are Read Multiple Times within a Function	SAKAI	🔵 Informational	Aknowledged
Sakai.sol:1376,1377,1383,1384	Variables Can Be Declared as Immutable	SAKAI	🔵 Informational	Aknowledged
Sakai.sol:1461	Get Contract Balance of ETH in Assembly	SAKAI	🔵 Informational	Aknowledged
Sakai.sol:450,971,972,1003,1064,1065,1460,1475,1489,1504	Use Assembly to Check Zero Address	Ownable	🔵 Informational	Aknowledged

Code Security - Unauthenticated Storage Access

Title	Severity	Location	Status
Unauthenticated Storage Access	● Medium	Sakai.sol:826	Aknowledged

Description

Modification to state variable(s) is not restricted by authenticating msg.sender.

Code Security - Unauthenticated Storage Access

Title	Severity	Location	Status
Unauthenticated Storage Access	● Medium	Sakai.sol:860	Aknowledged

Description

Modification to state variable(s) is not restricted by authenticating msg.sender.

Code Security - Unauthenticated Storage Access

Title	Severity	Location	Status
Unauthenticated Storage Access	● Medium	Sakai.sol:887	Aknowledged

Description

Modification to state variable(s) is not restricted by authenticating msg.sender.

Code Security - Unauthenticated Storage Access

Title	Severity	Location	Status
Unauthenticated Storage Access	● Medium	Sakai.sol:910	Aknowledged

Description

Modification to state variable(s) is not restricted by authenticating msg.sender.

Code Security - Use Safer Functions

Title	Severity	Location	Status
Use Safer Functions	● Low	Sakai.sol:1461	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 - Prefer .call() To send()/transfer()

Title	Severity	Location	Status
Prefer .call() To send()/transfer()	● Informational	Sakai.sol:1461	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	Sakai.sol:2,396,1373	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 - Unused Events

Title	Severity	Location	Status
Unused Events	● Informational	Sakai.sol:1138	Aknowledged

Description

Unused events increase contract size and gas usage at deployment.

Optimization Suggestion - No Check of Address Params with Zero Address

Title	Severity	Location	Status
No Check of Address Params with Zero Address	● Informational	Sakai.sol:1534,1551	Aknowledged

Description

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

Optimization Suggestion - Use Shift Operation Instead of Mul/Div

Title	Severity	Location	Status
Use Shift Operation Instead of Mul/Div	● Informational	Sakai.sol:1628	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 - Redundant Getter Function

Title	Severity	Location	Status
Redundant Getter Function	🔵 Informational	Sakai.sol:1564	Aknowledged

Description

A state variable with public visibility comes with a getter function to obtain the value of the variable, so there is no need to explicitly define a function to obtain the value of the variable. When deploying a contract, remove extra functions will save gas. About 37000 gas can be saved with optimization turned off while 6800 gas can be saved vice versa.

Optimization Suggestion - Function Visibility Can Be External

Title	Severity	Location	Status
Function Visibility Can Be External	🔵 Informational	Sakai.sol:910,934,1437,1547,1551,1564	Aknowledged

Description

Functions that are not called should be declared as external.

Optimization Suggestion - Use CustomError Instead of String

Title	Severity	Location	Status
Use CustomError Instead of String	🔵 Informational	Sakai.sol:430,449,687,941,971,972,977,1003,1064,1065,1086,1458,1470,1474,1484,1488,1498,1502,1503,1513,1517,1528,1538,1555,1660	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	Sakai.sol:1625	Aknowledged

Description

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

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	Sakai.sol:1430,1471,1485,1499	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 - Variables Can Be Declared as Immutable

Title	Severity	Location	Status
Variables Can Be Declared as Immutable	● Informational	Sakai.sol:1376,1377,1383,1384	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	Sakai.sol:1461	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.

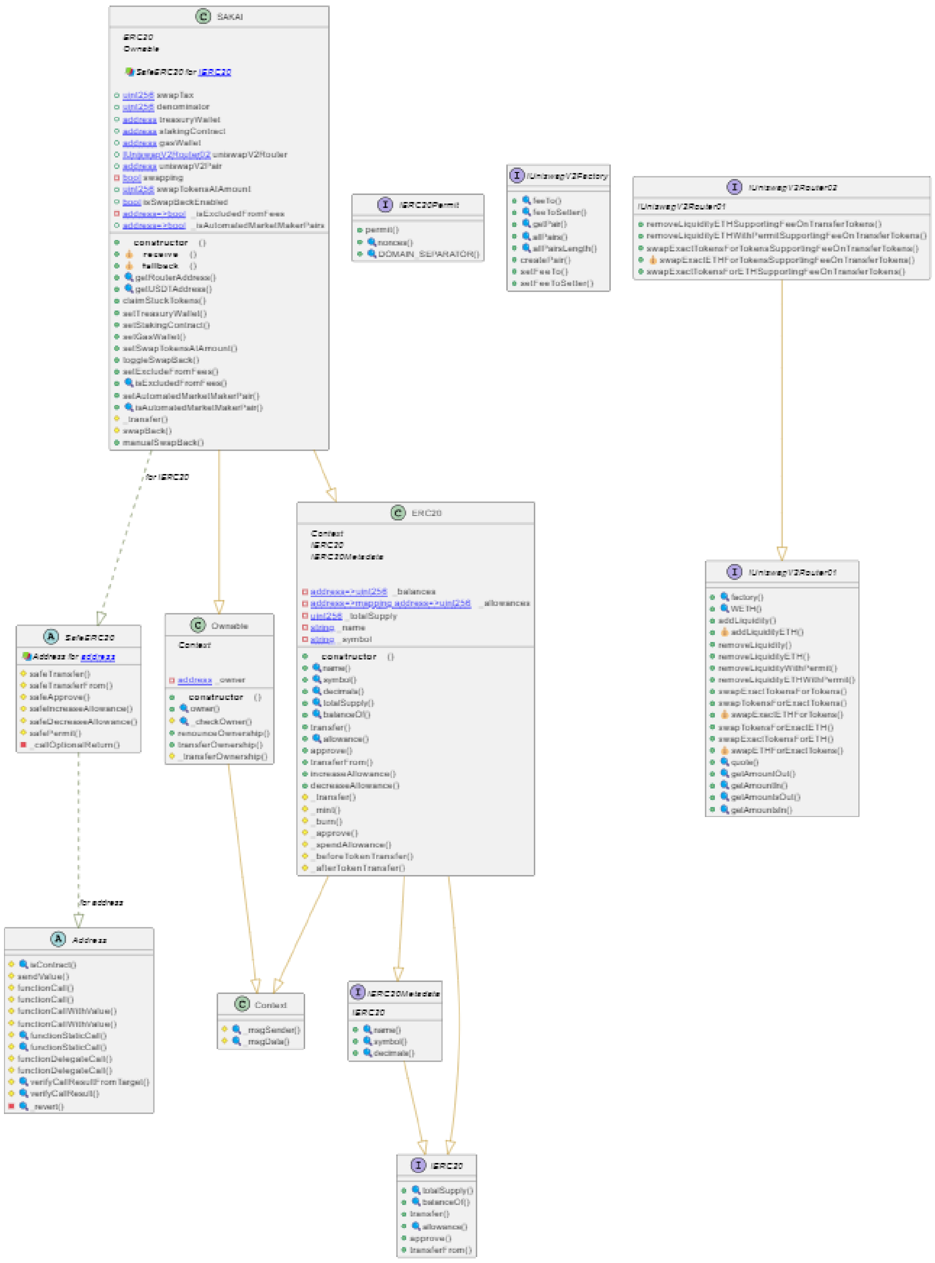
Optimization Suggestion - Use Assembly to Check Zero Address

Title	Severity	Location	Status
Use Assembly to Check Zero Address	🔵 Informational	Sakai.sol:450,971,972, 1003,1064,1065,1460, 1475,1489,1504	Aknowledged

Description

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

PlantUML


SAKAI
ERC20
Ownable
SafeERC20 for IERC20
uint256 swapTax
uint256 denominator
address treasuryWallet
address stakingContract
address gasWallet
IUniswapV2Router02 uniswapV2Router
address uniswapV2Pair
bool swapping
uint256 swapTokensAtAmount
bool isSwapBackEnabled
address=>bool _isExcludedFromFees
address=>bool _isAutomatedMarketMakerPairs
constructor ()
receive ()
fallback ()
getRouterAddress()
getUSDTAddress()
claimStuckTokens()
setTreasuryWallet()
setStakingContract()
setGasWallet()
setSwapTokensAtAmount()
toggleSwapBack()
setExcludeFromFees()
isExcludedFromFees()
setAutomatedMarketMakerPair()
isAutomatedMarketMakerPair()
_transfer()
swapBack()
manualSwapBack()
IERC20Permit
permit()
nonces()
DOMAIN_SEPARATOR()
IUniswapV2Factory
feeTo()
feeToSetter()
getPair()
allPairs()
allPairsLength()
createPair()
setFeeTo()
setFeeToSetter()
IUniswapV2Router02
IUniswapV2Router01
removeLiquidityETHSupportingFeeOnTransferTokens()
removeLiquidityETHWithPermitSupportingFeeOnTransferTokens()
swapExactTokensForTokensSupportingFeeOnTransferTokens()
swapExactETHForTokensSupportingFeeOnTransferTokens()
swapExactTokensForETHSupportingFeeOnTransferTokens()
for IERC20
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()
IUniswapV2Router01
factory()
WETH()
addLiquidity()
addLiquidityETH()
removeLiquidity()
removeLiquidityETH()
removeLiquidityWithPermit()
removeLiquidityETHWithPermit()
swapExactTokensForTokens()
swapTokensForExactTokens()
swapExactETHForTokens()
swapTokensForExactETH()
swapExactTokensForETH()
swapETHForExactTokens()
quote()
getAmountOut()
getAmountIn()
getAmountsOut()
getAmountsIn()
SafeERC20
Address for address
safeTransfer()
safeTransferFrom()
safeApprove()
safeIncreaseAllowance()
safeDecreaseAllowance()
safePermit()
_callOptionalReturn()
Ownable
Context
address _owner
constructor ()
owner()
_checkOwner()
renounceOwnership()
transferOwnership()
_transferOwnership()
for address
Address
isContract()
sendValue()
functionCall()
functionCall()
functionCallWithValue()
functionCallWithValue()
functionStaticCall()
functionStaticCall()
functionDelegateCall()
functionDelegateCall()
verifyCallResultFromTarget()
verifyCallResult()
_revert()
Context
_msgSender()
_msgData()
IERC20Metadata
IERC20
name()
symbol()
decimals()
IERC20
totalSupply()
balanceOf()
transfer()
allowance()
approve()
transferFrom()

Appendix

Finding Categories

Security and Best Practices

1. Unauthenticated Storage Access: Smart contracts should undergo scrutiny for unauthenticated storage access, which can lead to unauthorized data tampering.
2. Use Safer Functions: Utilize functions known for their secure design to mitigate potential security vulnerabilities. Review functions for enhanced security.
3. Prefer .call() To send()/transfer(): Employ .call() instead of send()/transfer() for external contract calls to minimize security risks.
4. Recommend to Follow Code Layout Conventions: Strict adherence to established code layout conventions can significantly improve code readability and maintainability.
5. Unused Events: Eliminate unused events to declutter the contract code and reduce the attack surface.
6. 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.
7. Use Shift Operation Instead of Mul/Div: Utilize shift operations instead of multiplication and division operations for improved gas efficiency and reduced risk of overflow/underflow.
8. Redundant Getter Function: Remove redundant getter functions to streamline the contract code and reduce gas consumption.
9. Function Visibility Can Be External: Enhance gas efficiency by setting functions to external visibility if they are accessible only from within the contract.
10. Use CustomError Instead of String: Opt for custom error codes instead of string error messages for more efficient contract operation.
11. ReentrancyGuard Should Modify External Function: Ensure that the ReentrancyGuard modifier is applied to functions modifying external state to prevent reentrancy attacks.
12. Cache State Variables That Are Read Multiple Times within a Function: Improve gas efficiency by caching state variables that are read multiple times within a function.
13. Variables Can Be Declared as Immutable: Declare variables as immutable if their values do not change after initialization to enhance security and readability.
14. Get Contract Balance of ETH in Assembly: Utilize assembly language to efficiently retrieve the contract's ETH balance for optimized gas usage.
15. Use Assembly to Check Zero Address: Employ optimized assembly checks to verify zero addresses efficiently and securely.

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://sakaivault.io/

Network Security

High | 0 Attentions

Application Security

High | 3 Attentions

DNS Security

Med | 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** | ℹ **4 Attention**

Missing SPF Record	YES ℹ
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

✓ 5 Passed | ⓘ 6 Failed

X (Twitter)	↗	PASS ✓
Facebook		FAIL ✗
Instagram		FAIL ✗
TikTok		FAIL ✗
YouTube		FAIL ✗
Twich		FAIL ✗
Telegram	↗	PASS ✓
Discord	↗	PASS ✓
Medium	↗	PASS ✓
Others	↗	PASS ✓

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

Evolving	**Medium-High**
Token Launch Date	**2023.03.14 14:00 (UTC)**
Token Market Cap (estimate)	**$8.16M**
Token/Project Age	**350 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	YES ✓
CoinGecko	YES ✓
Others	YES ✓

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

The coin is available on MEXC, Bitget, DigiFinex and Gate.io.

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.