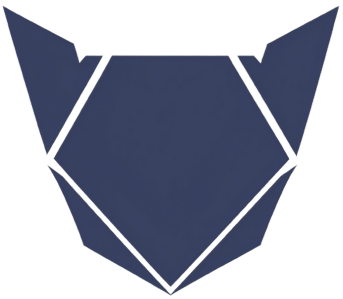


SPHINXSHIELD

Security Assessment

OpSec

Feb 28th, 2024

Evaluation Outcomes

Security Score

Review	Score
Overall Score	80/100
Auditor Score	80/100

Review by Section	Score
Manual Scan Score	48/57
Advance Check Score	14/19

Scoring System

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

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

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

Table of Contents

Summary

Overview

- Project Summary
- Audit Summary
- Vulnerability Summary
- Audit Scope

Understandings

Findings

PlantUML

Appendix

Website Scan

Social Media Checks

Fundamental Health

Coin Tracker Analytics

CEX Holding Analytics

Disclaimer

About

Summary

This audit report is tailored for **OpSec**, aiming to uncover potential issues and vulnerabilities within the **OpSec** 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	OpSec
Blockchain	Ethereum
Language	Solidity
Codebase	https://etherscan.io/token/0x6A7eFF1e2c355AD6eb91BEbB5ded49257F3FED98
Commit	8c244e7d147728813b8ea7a1428524563eca4b1b1ccd1d4d028800617cab6342

Audit Summary

Delivery Date	Feb 28th, 2024
Audit Methodology	Static Analysis, Manual Review
Key Components	OpSec.sol

Vulnerability Summary

24
Total Findings

0	0	0	24	0
Resolved	Mitigated	Partially Resolved	Aknowledged	Declined

Vulnerability Level	Total	Pending	Declined	Aknowledged	Resolved
High	2	0	0	2	0
Medium	6	0	0	6	0
Low	1	0	0	1	0
Informational	15	0	0	15	0
Discussion	0	0	0	0	0

Audit Scope

ID	File	KECCAK256 or SHA256 Checksum
OPC	OpSec.sol	0xede3e98a738433d5a1eb39a4df9c4bc2a507af9cc0371cc74477fce8c08e6523

Understandings

OpSec is an ERC-20 token with ownership functionality deployed on the Ethereum blockchain. The contract incorporates several features and mechanisms to manage its operations, including tax distribution, liquidity acquisition, and exclusive functions for modifying contract parameters.

Token Information

- Token Name: OpSec
- Symbol: OPSEC
- Decimals: 18
- Total Supply: 100,000,000 OPSEC

Fee Management

- The contract owner can configure various fees, including liquidity fee, team fee, marketing fee, dev fee, and burn fee, along with the fee denominator.
- Fee multipliers for buy, sell, and transfer transactions can be set by the owner.
- Addresses receiving fee components can be designated by the owner.

Tax Exemption

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

Ownership and Authorization

The contract owner can authorize specific addresses to access privileged functions, allowing configuration of contract and address attributes.

Transaction Limits

The contract imposes transaction limits to prevent excessive token movement, ensuring compliance with defined limits.

Swap Mechanism

OpSec utilizes a swap mechanism to manage liquidity, swapping tokens to ETH tokens via the Uniswap Router when a set threshold is reached.

Open Trading

Trading can be restricted based on conditions set by the owner until specific requirements are met.

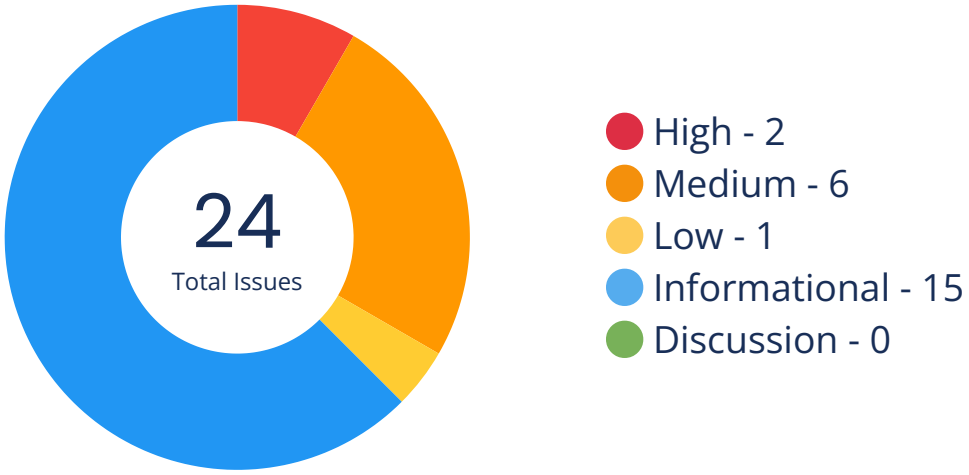
Additional Functionality

The contract includes functions for clearing stuck ETH, clearing tokens, and other supplementary operations.

About OpSec

OpSec aims to create a secure, efficient, and decentralized digital ecosystem. Leveraging advanced AI technology, OpSec builds, maintains, and operates blockchain infrastructure to ensure the security and privacy of blockchain applications.

Findings



Location	Title	Scope	Severity	Status
OpSec.sol:637	Freeze Money	OpSec	High	Aknowledged
OpSec.sol:671	Unchecked Return Value of ecrecover	OpSec	High	Aknowledged
OpSec.sol:646	Weak Sources of Randomness	OpSec	Medium	Aknowledged
OpSec.sol:116	Unauthenticated Storage Access	ERC20	Medium	Aknowledged
OpSec.sol:125	Unauthenticated Storage Access	ERC20	Medium	Aknowledged
OpSec.sol:130	Unauthenticated Storage Access	ERC20	Medium	Aknowledged
OpSec.sol:146	Unauthenticated Storage Access	ERC20	Medium	Aknowledged
OpSec.sol:644	Unauthenticated Storage Access	OpSec	Medium	Aknowledged

Location	Title	Scope	Severity	Status
OpSec.sol:539,564,565	Uninitialized Variables	OpSec	Low	Aknowledged
OpSec.sol:537	Set the Constant to Private	OpSec	Informational	Aknowledged
OpSec.sol:3,15,49,361,532	Recommend to Follow Code Layout Conventions	Ownable	Informational	Aknowledged
OpSec.sol:332,362,367,410,411,417,425,573,793	Unused Events	IUniswapV2Factory	Informational	Aknowledged
OpSec.sol:724,756,761,779,784	No Check of Address Params with Zero Address	OpSec	Informational	Aknowledged
OpSec.sol:728,757	Continuous State Variable Write	OpSec	Informational	Aknowledged
OpSec.sol:898	Use != 0 Instead of > 0 for Unsigned Integer Comparison	OpSec	Informational	Aknowledged
OpSec.sol:146,151,724,756,761,789	Function Visibility Can Be External	ERC20	Informational	Aknowledged
OpSec.sol:3	Floating Pragma	Global	Informational	Aknowledged
OpSec.sol:29,38,138,153,166,167,172,184,217,218,672,696,700,709,717,743,753,765,897,898	Use CustomError Instead of String	Ownable	Informational	Aknowledged
OpSec.sol:897	Lack of Error Message	OpSec	Informational	Aknowledged
OpSec.sol:599	Cache State Variables That Are Read Multiple Times within a Function	OpSec	Informational	Aknowledged

Location	Title	Scope	Severity	Status
OpSec.sol:536	Variables Can Be Declared as Immutable	OpSec	Informational	Aknowledged
OpSec.sol:38,166,167,184,217,218	Use Assembly to Check Zero Address	Ownable	Informational	Aknowledged
OpSec.sol:697	Too Many Digits	OpSec	Informational	Aknowledged
OpSec.sol:671	Check Risk of Transaction Replay	OpSec	Informational	Aknowledged

Code Security - Freeze Money

Title	Severity	Location	Status
Freeze Money	High	OpSec.sol:637	Aknowledged

Description

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

Code Security - Unchecked Return Value of ecrecover

Title	Severity	Location	Status
Unchecked Return Value of ecrecover	High	OpSec.sol:671	Aknowledged

Description

When using the ecrecover for signature verification, if an invalid signature is passed in, ecrecover will return a zero address. If the zero address check is not performed on the return value of ecrecover, wrong business logic may occur.

Code Security - Weak Sources of Randomness

Title	Severity	Location	Status
Weak Sources of Randomness	Medium	OpSec.sol:646	Aknowledged

Description

You are likely only to use chain attributes(e.g. block.timestamp, blockhash) as random seed which is insecure, as miner/consensus node could intentionally control these values.

Code Security - Unauthenticated Storage Access

Title	Severity	Location	Status
Unauthenticated Storage Access	Medium	OpSec.sol:116	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	OpSec.sol:125	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	OpSec.sol:130	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	OpSec.sol:146	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	OpSec.sol:644	Aknowledged

Description

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

Code Security - Uninitialized Variables

Title	Severity	Location	Status
Uninitialized Variables	Low	OpSec.sol:539,564,565	Aknowledged

Description

Variables that are not initialized after definition are used in the contract.

Optimization Suggestion - Set the Constant to Private

Title	Severity	Location	Status
Set the Constant to Private	Informational	OpSec.sol:537	Aknowledged

Description

For constants, if the visibility is set to public, the compiler will automatically generate a getter function for it, which will consume more gas during deployment.

Optimization Suggestion - Recommend to Follow Code Layout Conventions

Title	Severity	Location	Status
Recommend to Follow Code Layout Conventions	Informational	OpSec.sol:3,15,49,361, 532	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	OpSec.sol:332,362,367 ,410,411,417,425,573, 793	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	OpSec.sol:724,756,761,779,784	Aknowledged

Description

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

Optimization Suggestion - Continuous State Variable Write

Title	Severity	Location	Status
Continuous State Variable Write	Informational	OpSec.sol:728,757	Aknowledged

Description

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

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

Title	Severity	Location	Status
Use != 0 Instead of > 0 for Unsigned Integer Comparison	Informational	OpSec.sol:898	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	OpSec.sol:146,151,724,756,761,789	Aknowledged

Description

Functions that are not called should be declared as external.

Optimization Suggestion - Floating Pragma

Title	Severity	Location	Status
Floating Pragma	Informational	OpSec.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 - Use CustomError Instead of String

Title	Severity	Location	Status
Use CustomError Instead of String	Informational	OpSec.sol:29,38,138,153,166,167,172,184,217,218,672,696,700,709,717,743,753,765,897,898	Aknowledged

Description

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

Optimization Suggestion - Lack of Error Message

Title	Severity	Location	Status
Lack of Error Message	Informational	OpSec.sol:897	Aknowledged

Description

Use empty string as parameter while invoking function revert or require.

Optimization Suggestion - Variables Can Be Declared as Immutable

Title	Severity	Location	Status
Variables Can Be Declared as Immutable	Informational	OpSec.sol:536	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 - Use Assembly to Check Zero Address

Title	Severity	Location	Status
Use Assembly to Check Zero Address	Informational	OpSec.sol:38,166,167, 184,217,218	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	OpSec.sol:697	Aknowledged

Description

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

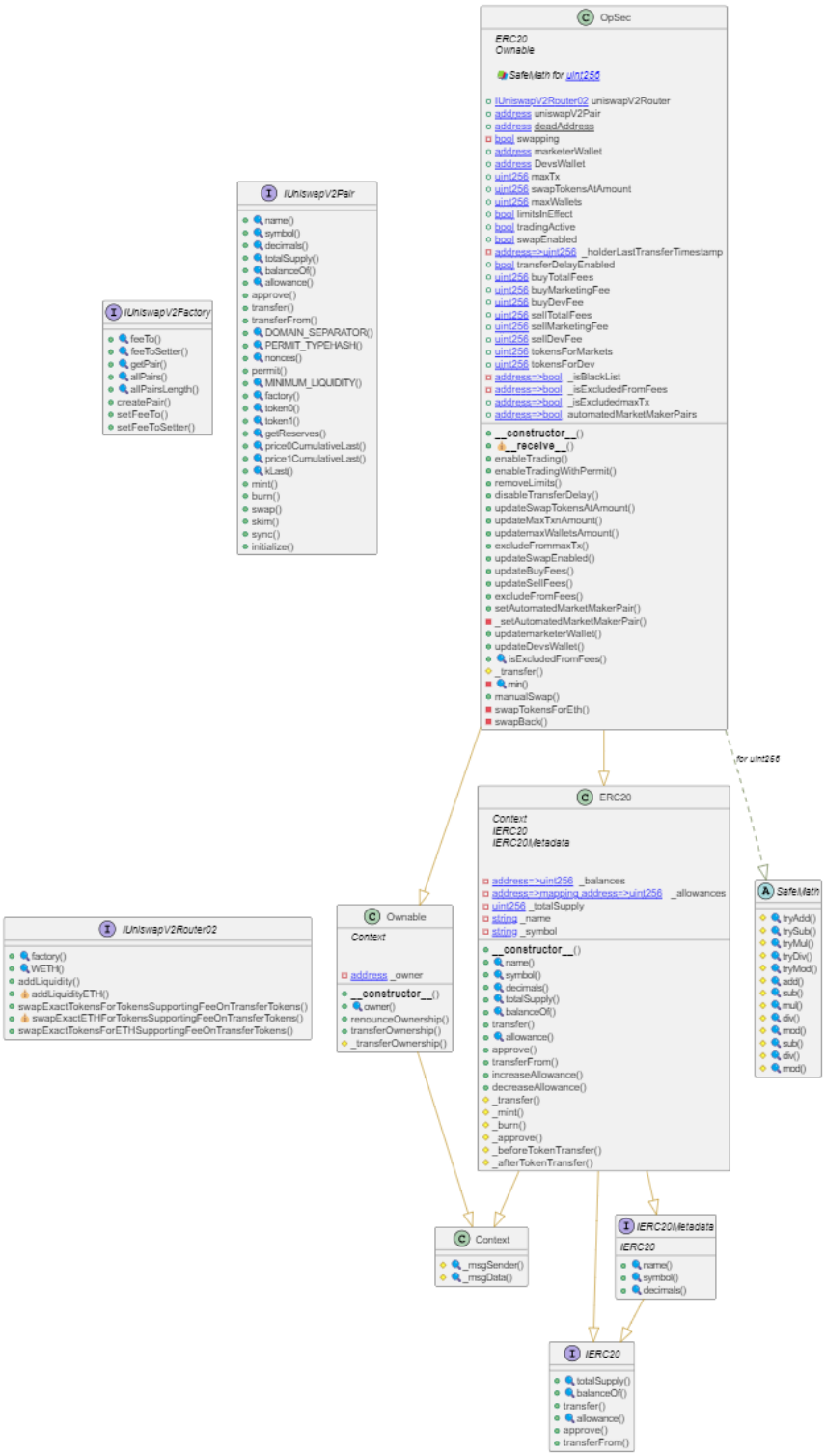
Optimization Suggestion - Check Risk of Transaction Replay

Title	Severity	Location	Status
Check Risk of Transaction Replay	Informational	OpSec.sol:671	Aknowledged

Description

Be aware of transaction replay risks when using signatures.

PlantUML


OpSec
ERC20
Ownable
SafeMath for uint256
IUniswapV2Router02 uniswapV2Router
address uniswapV2Pair
address deadAddress
bool swapping
address marketerWallet
address DevsWallet
uint256 maxTx
uint256 swapTokensAtAmount
uint256 maxWallets
bool limitsInEffect
bool tradingActive
bool swapEnabled
address=>uint256 _holderLastTransferTimestamp
bool transferDelayEnabled
uint256 buyTotalFees
uint256 buyMarketingFee
uint256 buyDevFee
uint256 sellTotalFees
uint256 sellMarketingFee
uint256 sellDevFee
uint256 tokensForMarkets
uint256 tokensForDev
address=>bool _isBlackList
address=>bool _isExcludedFromFees
address=>bool _isExcludedmaxTx
address=>bool automatedMarketMakerPairs
__constructor__()
__receive__()
enableTrading()
enableTradingWithPermit()
removeLimits()
disableTransferDelay()
updateSwapTokensAtAmount()
updateMaxTxnAmount()
updatemaxWalletsAmount()
excludeFrommaxTx()
updateSwapEnabled()
updateBuyFees()
updateSellFees()
excludeFromFees()
setAutomatedMarketMakerPair()
_setAutomatedMarketMakerPair()
updatemarketerWallet()
updateDevsWallet()
isExcludedFromFees()
_transfer()
min()
manualSwap()
swapTokensForEth()
swapBack()
IUniswapV2Pair
name()
symbol()
decimals()
totalSupply()
balanceOf()
allowance()
approve()
transfer()
transferFrom()
DOMAIN_SEPARATOR()
PERMIT_TYPEHASH()
nonces()
permit()
MINIMUM_LIQUIDITY()
factory()
token0()
token1()
getReserves()
price0CumulativeLast()
price1CumulativeLast()
kLast()
mint()
burn()
swap()
skim()
sync()
initialize()
IUniswapV2Factory
feeTo()
feeToSetter()
getPair()
allPairs()
allPairsLength()
createPair()
setFeeTo()
setFeeToSetter()
for uint256
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()
_beforeTokenTransfer()
_afterTokenTransfer()
SafeMath
tryAdd()
trySub()
tryMul()
tryDiv()
tryMod()
add()
sub()
mul()
div()
mod()
sub()
div()
mod()
Ownable
Context
address _owner
__constructor__()
owner()
renounceOwnership()
transferOwnership()
_transferOwnership()
IUniswapV2Router02
factory()
WETH()
addLiquidity()
addLiquidityETH()
swapExactTokensForTokensSupportingFeeOnTransferTokens()
swapExactETHForTokensSupportingFeeOnTransferTokens()
swapExactTokensForETHSupportingFeeOnTransferTokens()
Context
_msgSender()
_msgData()
IERC20Metadata
IERC20
name()
symbol()
decimals()
IERC20
totalSupply()
balanceOf()
transfer()
allowance()
approve()
transferFrom()

Appendix

Finding Categories

Security and Best Practices

1. Freeze Money: Smart contracts should implement mechanisms to freeze funds in case of emergencies or security breaches to prevent unauthorized withdrawals.
2. Unchecked Return Value of ecrecover: Careful consideration should be given to the return value of the ecrecover function to ensure that it is properly validated before being used in critical operations.
3. Weak Sources of Randomness: Avoid using weak sources of randomness, such as block timestamps, for generating random values as they can be manipulated by miners.
4. Unauthenticated Storage Access: Smart contracts should undergo scrutiny for unauthenticated storage access, which can lead to unauthorized data tampering.
5. Uninitialized Variables: All variables should be properly initialized to prevent unexpected behavior and vulnerabilities arising from uninitialized state.
6. Set the Constant to Private: Declared constants should be set to private visibility to prevent unwanted external access and modification.
7. Recommend to Follow Code Layout Conventions: Strict adherence to established code layout conventions can significantly improve code readability, maintainability, and ease of review.
8. Unused Events: Remove unused events from smart contracts to reduce contract size and potential confusion during contract analysis.
9. 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 to prevent unintended behavior.
10. Continuous State Variable Write: Minimize continuous state variable writes to reduce gas costs and optimize contract efficiency.
11. Use != 0 Instead of > 0 for Unsigned Integer Comparison: Prefer using the != (not equal) operator instead of > (greater than) for unsigned integer comparison to avoid potential edge cases.
12. Function Visibility Can Be External: Enhance gas efficiency by setting functions to external visibility if they are accessible only from within the contract.
13. Floating Pragma: Ensure that your Solidity pragma remains consistent for added contract security and compatibility with different compiler versions.
14. Use CustomError Instead of String: Opt for custom error codes instead of string error messages for more efficient contract operation and reduced gas costs.
15. Lack of Error Message: Provide meaningful error messages in revert/require statements to assist developers and users in understanding contract failures.
16. Cache State Variables That Are Read Multiple Times within a Function: Cache frequently read state variables within functions to optimize gas usage and improve contract performance.
17. Variables Can Be Declared as Immutable: Declare variables as immutable if they do not change after initialization to enhance security and readability.
18. Use Assembly to Check Zero Address: Utilize optimized assembly checks to verify zero addresses efficiently and securely within smart contracts.
19. Too Many Digits: Avoid using excessively large or precise numeric values to minimize gas costs and contract complexity.
20. Check Risk of Transaction Replay: Evaluate the risk of transaction replay attacks and implement appropriate safeguards to mitigate potential vulnerabilities.

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://opsec.software/

Network Security

High | 0 Attentions

Application Security

High | 3 Attentions

DNS Security

High | 3 Attentions

Network Security

9 Passed **0 Attention**

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

Application Security

8 Passed | 3 Attention

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

DNS Health

7 Passed | 3 Attention

Missing SPF Record	YES
Missing DMARC Record	NO
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

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

Emerging	**Low**
Token Launch Date	**2024.01.03 00:00 (UTC)**
Token Market Cap (estimate)	**$116.42M**
Token/Project Age	**57 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.

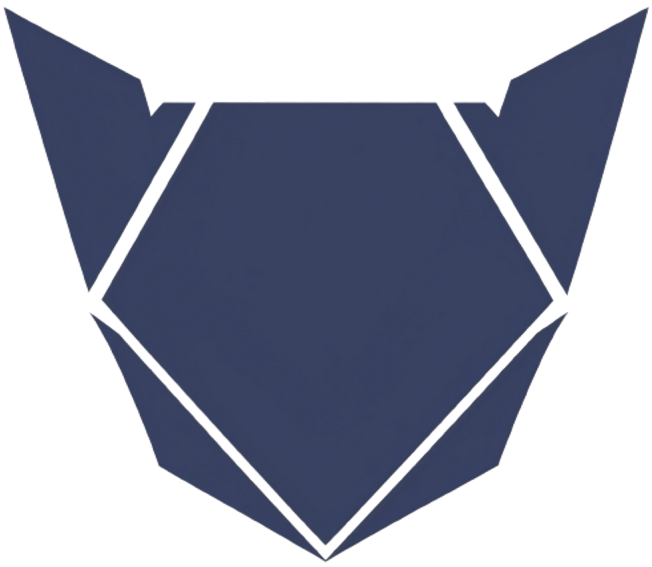
Coin Tracker Analytics

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 OpenOcean, ExMarkets and CoinEx.

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.