

SPHINXSHIELD

Security Assessment

Operation Phoenix

Feb 4th, 2024

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



Evaluation Outcomes

Security Score

Review	Score
Overall Score	84/100
Auditor Score	80/100

Review by Section	Score
Manual Scan Score	46/57
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 **Operation Phoenix**, aiming to uncover potential issues and vulnerabilities within the **Operation Phoenix** 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

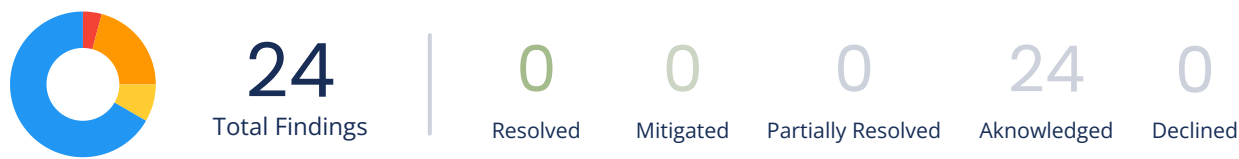
Project Summary

Project Name	Operation Phoenix
Blockchain	Binance Smart Chain
Language	Solidity
Codebase	https://bscscan.com/token/0x59803e5fe213d4b22fb9b061c4c89e716a1ca760
Commit	73ada993e696148609a9aec0952a95aea0105ce574864a0d72ba0f5a13bb7bad

Audit Summary

Delivery Date	Feb 4th, 2024
Audit Methodology	Static Analysis, Manual Review
Key Components	OperationPhoenix.sol

Vulnerability Summary



Vulnerability Level	Total	⚠ Pending	⊗ Declined	ℹ Acknowledged	✅ Resolved
High	1	0	0	1	0
Medium	5	0	0	5	0
Low	2	0	0	2	0
Informational	16	0	0	16	0
Discussion	0	0	0	0	0



Audit Scope

ID	File	KECCAK256 or SHA256 Checksum
OPX	OperationPhoenix.sol	0x23fcb7ee627e7aa047a3752ae2061d723574475d63f778c5fd4a8858cde60a6d



Understandings

Operation Phoenix is a BSC-based ERC20 token with unique mechanisms for managing operations. Here's an overview of its key components:

Token Information

- Token Name: Operation Phoenix
- Symbol: \$OPHX
- Decimals: 18
- Total Supply: 100,000,000 OPHX

Fee Distribution

- Liquidity Fee: Collected for providing liquidity, owner-adjustable.
- Team Fee: Allocated to the project team, owner-adjustable.
- Marketing Fee: Dedicated to marketing efforts, owner-adjustable.
- Dev Fee: Allocated for development purposes, owner-adjustable.
- Burn Fee: Tokens burned to reduce total supply, owner-adjustable.
- Total Fee: Sum of the above fees, used for overall fee distribution.
- Fee Denominator: Denominator in fee calculations, usually set to 100.

Fee Management

- Set Tax: Owner-configurable liquidity, team, marketing, dev, and burn fees.
- Set Fee Multipliers: Adjust percentage fees for buy, sell, and transfer transactions.
- Set Fee Receivers: Owner-defined addresses for receiving fee components.

Tax Exemption

Owner can exempt specific addresses from fees, allowing fee exemption for whitelisted wallets or contracts.

Ownership and Authorization

Contract owner can authorize specific addresses with privileged functions. Functions are restricted by the onlyOwner modifier, used for configuring the contract and address attributes.



Transaction Limits

Enforces transaction limits to prevent excessive token movement.

Swap Mechanism

- Utilizes a swap mechanism for liquidity management.
- Swaps tokens to BB tokens via PancakeSwap Router, affecting the token's price temporarily.
- Remaining balance is supplied to the OPHX-BNB liquidity pool.

Open Trading

Trading can be restricted based on conditions defined by the owner.

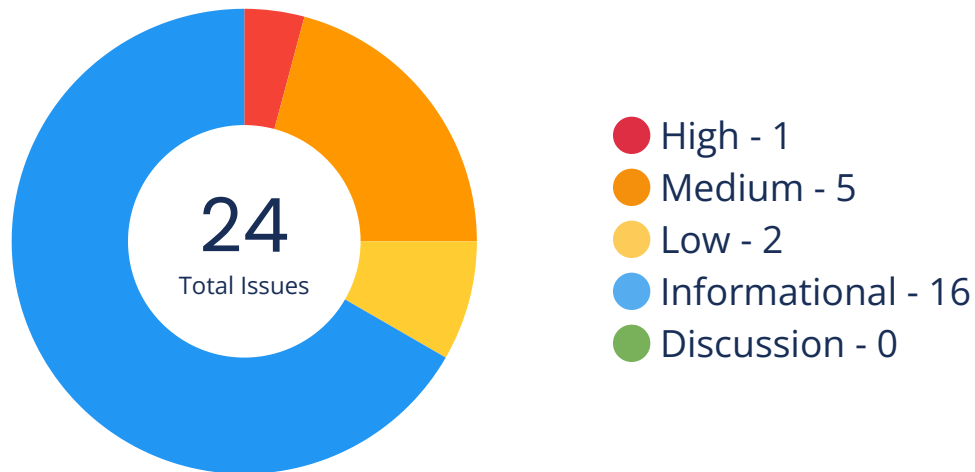
Additional Functionality

Includes functions for clearing stuck ETH, clearing tokens, and more.

Operation Phoenix is a BSC-based project aiming to fulfill the potential that SafeMoon was expected to achieve. The Operation Phoenix contract, an ERC20 token on the BSC chain, incorporates innovative features for secure and efficient operations.



Findings



Location	Title	Scope	Severity	Status
OperationPhoenix.sol:1159	Reentrancy	OperationPhoenix	High	Acknowledged
OperationPhoenix.sol:1189	Unchecked Call Return Value	OperationPhoenix	Medium	Acknowledged
OperationPhoenix.sol:271	Unauthenticated Storage Access	ERC20	Medium	Acknowledged
OperationPhoenix.sol:294	Unauthenticated Storage Access	ERC20	Medium	Acknowledged
OperationPhoenix.sol:316	Unauthenticated Storage Access	ERC20	Medium	Acknowledged
OperationPhoenix.sol:339	Unauthenticated Storage Access	ERC20	Medium	Acknowledged
OperationPhoenix.sol:982	Use Safer Functions	OperationPhoenix	Low	Acknowledged
OperationPhoenix.sol:898	Uninitialized Variables	OperationPhoenix	Low	Acknowledged



Location	Title	Scope	Severity	Status
OperationPhoenix. sol:982	Prefer .call() To send()/transfer()	OperationPh oenix	● Informational	Acknowledged
OperationPhoenix. sol:2,36,756,891	Recommend to Follow Code Layout Conventions	Ownable	● Informational	Acknowledged
OperationPhoenix. sol:1199,1200	Parameters Should Be Declared as Calldata	OperationPh oenix	● Informational	Acknowledged
OperationPhoenix. sol:757,758,775,77 7,921,923,929,932	Unused Events	IUniswapV2 Pair	● Informational	Acknowledged
OperationPhoenix. sol:969,975,995	No Check of Address Params with Zero Address	OperationPh oenix	● Informational	Acknowledged
OperationPhoenix. sol:897	Variables Should Be Constants	OperationPh oenix	● Informational	Acknowledged
OperationPhoenix. sol:891	No Need To Use SafeMath in Solidity Contract of Version 0.8.0 and Above	OperationPh oenix	● Informational	Acknowledged
OperationPhoenix. sol:1204	Use ++i/--i Instead of i++/i--	OperationPh oenix	● Informational	Acknowledged
OperationPhoenix. sol:339,359,975,98 5,989,995,1005,102 1,1159	Function Visibility Can Be External	ERC20	● Informational	Acknowledged
OperationPhoenix. sol:67,86,362,389,3 90,395,416,473,474 ,495,964,990,1016, 1202,1203	Use CustomError Instead of String	Ownable	● Informational	Acknowledged
OperationPhoenix. sol:964,990	Lack of Error Message	OperationPh oenix	● Informational	Acknowledged
OperationPhoenix. sol:896	Unused State Variables	OperationPh oenix	● Informational	Acknowledged



Location	Title	Scope	Severity	Status
OperationPhoenix. sol:1013,1014,1161 ,1180	Cache State Variables That Are Read Multiple Times within a Function	OperationPh oenix	● Informational	Aknowledged
OperationPhoenix. sol:894	Variables Can Be Declared as Immutable	OperationPh oenix	● Informational	Aknowledged
OperationPhoenix. sol:981	Get Contract Balance of ETH in Assembly	OperationPh oenix	● Informational	Aknowledged
OperationPhoenix. sol:86,389,390,416, 473,474	Use Assembly to Check Zero Address	Ownable	● Informational	Aknowledged



Code Security – Reentrancy

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

Reentrancy

● High

OperationPhoenix.sol:
1159

Aknowledged

Description

As the external call is executed prior to state variables alterations, it exposes the possibility for the external contract to perform a reentrancy attack by calling back into this contract.

Code Security – Unchecked Call Return Value

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

Unchecked Call Return Value

● Medium

OperationPhoenix.sol:
1189

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 – Unauthenticated Storage Access

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

Unauthenticated Storage Access

● Medium

OperationPhoenix.sol:
271

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

OperationPhoenix.sol:
294

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

OperationPhoenix.sol:
316

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

OperationPhoenix.sol:
339

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

OperationPhoenix.sol:
982

Acknowledged

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.

Code Security – Uninitialized Variables

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

Uninitialized Variables

● Low

OperationPhoenix.sol:
898

Acknowledged

Description

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

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

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

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

● Informational

OperationPhoenix.sol:
982

Acknowledged

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

OperationPhoenix.sol:
2,36,756,891

Acknowledged

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

OperationPhoenix.sol:
1199,1200

Acknowledged

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 - Unused Events

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

Unused Events

● Informational

OperationPhoenix.sol:
757,758,775,777,921,923,929,932

Acknowledged

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	OperationPhoenix.sol: 969,975,995	Aknowledged

Description

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

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

Title	Severity	Location	Status
No Need To Use SafeMath in Solidity Contract of Version 0.8.0 and Above	● Informational	OperationPhoenix.sol: 891	Aknowledged

Description

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

Optimization Suggestion – Use ++i/--i Instead of i++/i--

Title	Severity	Location	Status
Use ++i/--i Instead of i++/i--	● Informational	OperationPhoenix.sol: 1204	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 – Function Visibility Can Be External

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

Function Visibility Can Be External

● Informational

OperationPhoenix.sol:
339,359,975,985,989,9
95,1005,1021,1159

Acknowledged

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

OperationPhoenix.sol:
67,86,362,389,390,395
,416,473,474,495,964,
990,1016,1202,1203

Acknowledged

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

OperationPhoenix.sol:
964,990

Acknowledged

Description

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



Optimization Suggestion – Unused State Variables

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

Unused State Variables

● Informational

OperationPhoenix.sol:
896

Aknowledged

Description

The state variables are not used, which will increase the gas consumption.

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

OperationPhoenix.sol:
1013,1014,1161,1180

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

OperationPhoenix.sol:
894

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	OperationPhoenix.sol: 981	Acknowledged

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	OperationPhoenix.sol: 86,389,390,416,473,474	Acknowledged

Description

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

[illegible]



Appendix

Finding Categories

Security and Best Practices

1. Reentrancy: Guard against reentrancy attacks by ensuring that external calls are made after state changes.
2. Unchecked Call Return Value: Always check return values of external calls to prevent unexpected behavior.
3. Unauthenticated Storage Access: Scrutinize contracts for unauthenticated storage access to prevent unauthorized data tampering.
4. Use Safer Functions: Employ functions with secure design to mitigate potential security vulnerabilities.
5. Uninitialized Variables: Avoid using uninitialized variables to prevent unexpected behavior and vulnerabilities.
6. Prefer .call() To send()/transfer(): Minimize security risks by using .call() for external contract calls.
7. Recommend to Follow Code Layout Conventions: Adhere to established code layout conventions to enhance readability and maintainability.
8. Parameters Should Be Declared as Calldata: Declare parameters as calldata to optimize gas usage and prevent unintended modifications.
9. Unused Events: Remove unused events to declutter contracts and improve readability.
10. No Check of Address Params with Zero Address: Verify address parameters to ensure they are not the zero address.
11. Variables Should Be Constants: Declare variables as constants when their values do not change after initialization.
12. No Need To Use SafeMath in Solidity Contract of Version 0.8.0 and Above: Leverage built-in overflow and underflow protection in Solidity versions 0.8.0 and above.
13. Use ++i/--i Instead of i++/i--: Enhance gas efficiency by using pre-increment/decrement operators.
14. Function Visibility Can Be External: Set function visibility to external if they are only accessible from within the contract.
15. Use CustomError Instead of String: Opt for custom error codes for efficient contract operation.
16. Lack of Error Message: Provide clear error messages to aid in debugging and user understanding.
17. Unused State Variables: Remove unused state variables to reduce contract complexity and potential attack surface.
18. Cache State Variables That Are Read Multiple Times within a Function: Improve efficiency by caching state variables read multiple times within a function.
19. Variables Can Be Declared as Immutable: Declare variables as immutable if they do not change after initialization.
20. Get Contract Balance of ETH in Assembly: Employ assembly to retrieve the contract's ETH balance efficiently.
21. Use Assembly to Check Zero Address: Utilize assembly for optimized zero address checks.



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://operationphoenix.network/>



Network Security

High | 0 Attentions

Application Security

Med | 6 Attentions










DNS Security

High | 2 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



5 Passed



6 Attention

Missing X-Frame-Options Header

YES

Missing HSTS header

YES

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

YES

Support SSL Protocols

NO

Support TLS Weak Version

YES



DNS Health



8 Passed



2 Attention

Missing SPF Record

NO



Missing DMARC Record

YES



Missing DKIM Record

NO



Ineffective SPF Record

NO



SPF Record Contains a Softfail Without DMARC

NO



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

YES





Social Media Checks

✓ 5 Passed

i 5 Failed

X (Twitter)



PASS ✓

Facebook

FAIL ✗

Instagram

FAIL ✗

TikTok

FAIL ✗

YouTube



PASS ✓

Twich

FAIL ✗

Telegram



PASS ✓

Discord



PASS ✓

Medium

FAIL ✗

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

Emerging

LOW

Token Launch Date

2024.02.01 19:00 (UTC)

Token Market Cap (estimate)

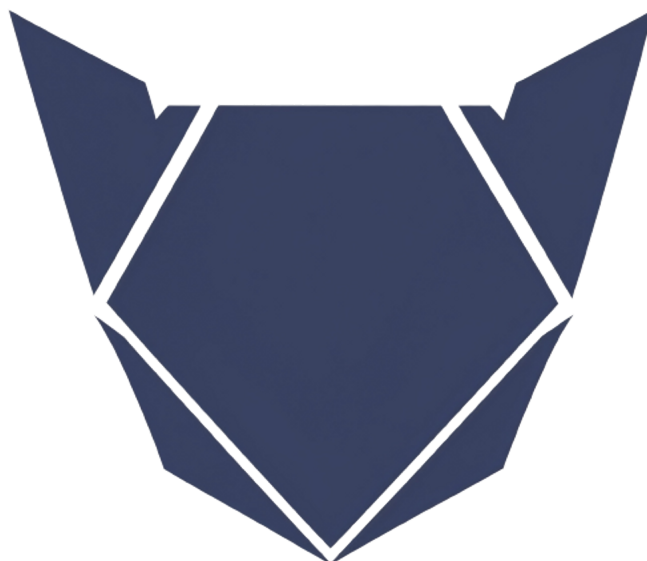
\$5.16M

Token/Project Age

8 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

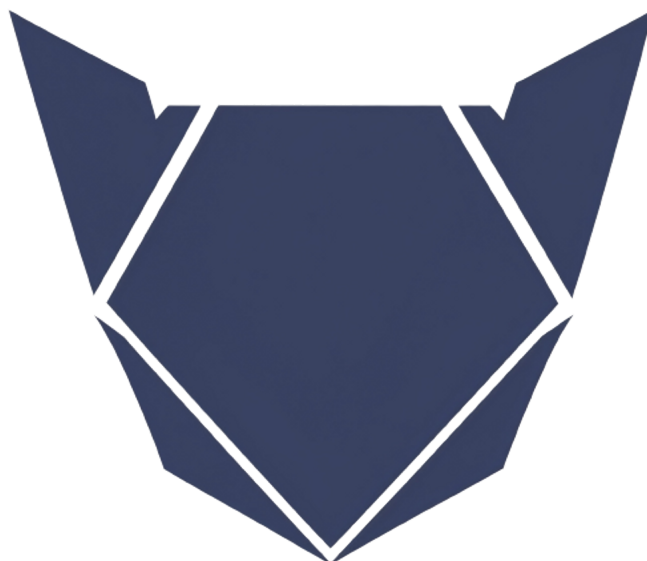
NO 

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.

