

SPHINXSHIELD

Security Assessment

StepGO

Mar 3th, 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	86/100
Auditor Score	82/100

Review by Section	Score
Manual Scan Score	50/57
Advance Check Score	16/19

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 **StepGO**, aiming to uncover potential issues and vulnerabilities within the **StepGO** 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	StepGO
Blockchain	Binance Smart Chain
Language	Solidity
Codebase	https://bscscan.com/token/0x30cf1bc18071a42b3c6df70e657383068cb64241
Commit	53f2f9183fe89978c801f051cd96d911feb83c596471d6260c776aa960049c7e

Audit Summary

Delivery Date	Mar 3th, 2024
Audit Methodology	Static Analysis, Manual Review
Key Components	STEPGO.sol

Vulnerability Summary



22
Total Findings

0
Resolved

0
Mitigated

0
Partially Resolved

22
Acknowledged

0
Declined

Vulnerability Level	Total	⚠ Pending	⊗ Declined	ℹ Acknowledged	✅ Resolved
High	0	0	0	0	0
Medium	5	0	0	5	0
Low	2	0	0	2	0
Informational	15	0	0	15	0
Discussion	0	0	0	0	0



Audit Scope

ID	File	KECCAK256 or SHA256 Checksum
STG	STEPGO.sol	0xb9a611d4f54f16fe391a592f6ea6400892b8fbaf29093e503b646fc788ec22fb



Understandings

StepGO is an ERC20 token deployed on the Binance Smart Chain (BSC) network. Here's a breakdown of its key components and functionalities:

Token Information

- Token Name: STEP GO
- Symbol: GO
- Decimals: 18
- Total Supply: 100,000,000,000 GO

Fee Management

- Marketing Fee: 4% on buy and sell transactions.
- Marketing Wallet: Address where marketing fees are sent.
- Maximum Wallet Amount: Limited to 3% of the total supply.
- Fee Exclusion: Specific addresses can be excluded from fees.
- Fee Adjustment: Owners can adjust marketing fees and development wallet.

Ownership and Authorization

- Contract Owner: Manages privileged functions.
- Development Wallet: Address used for development purposes.
- Privileged Functions: Restricted by the onlyOwner modifier.

Transaction Limits

Max Wallet Limit: Enforced to prevent excessive token accumulation.

Swap Mechanism

- Automated Swaps: Triggered when contract balance meets set threshold.
- Liquidity Management: Tokens are swapped for BNB and sent to the marketing wallet.



Additional Functionality

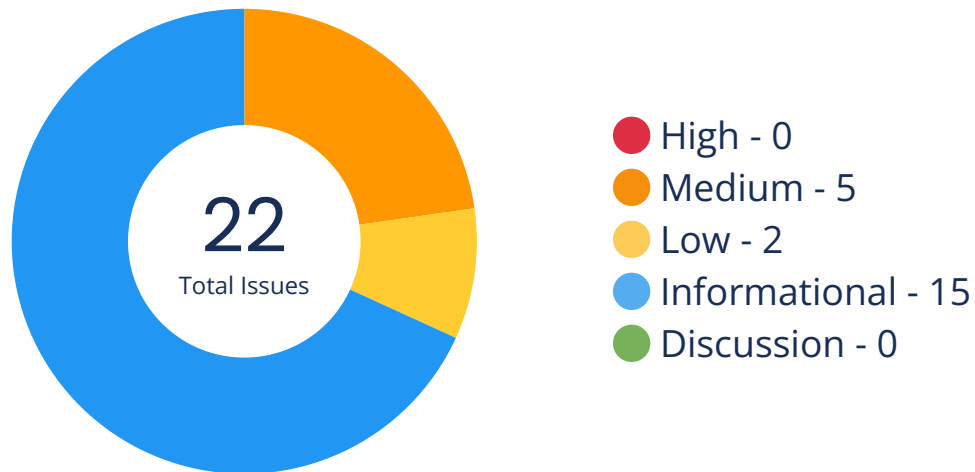
- Stuck Tokens: Owner can claim tokens stuck in the contract.
- Excluded Addresses: Certain addresses are excluded from fees and wallet limits.

About OpSec

StepGO operates on the Binance Smart Chain, offering fee management, ownership control, and liquidity management features to enhance its functionality and security.



Findings



Location	Title	Scope	Severity	Status
STEPGO.sol:493	Unchecked Call Return Value	STEPGO	Medium	Aknowledged
STEPGO.sol:106	Unauthenticated Storage Access	ERC20	Medium	Aknowledged
STEPGO.sol:115	Unauthenticated Storage Access	ERC20	Medium	Aknowledged
STEPGO.sol:120	Unauthenticated Storage Access	ERC20	Medium	Aknowledged
STEPGO.sol:138	Unauthenticated Storage Access	ERC20	Medium	Aknowledged
STEPGO.sol:488,493	Use Safer Functions	STEPGO	Low	Aknowledged
STEPGO.sol:442	Uninitialized Variables	STEPGO	Low	Aknowledged
STEPGO.sol:488	Prefer .call() To send()/transfer()	STEPGO	Informational	Aknowledged



Location	Title	Scope	Severity	Status
STEPGO.sol:2,4,37,242,428	Recommend to Follow Code Layout Conventions	IERC20	● Informational	Acknowledged
STEPGO.sol:230,243,244,263,264,265,273,451	Unused Events	IUniswapV2Factory	● Informational	Acknowledged
STEPGO.sol:507,518	No Check of Address Params with Zero Address	STEPGO	● Informational	Acknowledged
STEPGO.sol:440	Variables Should Be Constants	STEPGO	● Informational	Acknowledged
STEPGO.sol:138,143,514,523	Function Visibility Can Be External	ERC20	● Informational	Acknowledged
STEPGO.sol:2	Floating Pragma	Global	● Informational	Acknowledged
STEPGO.sol:51,60,127,145,158,159,164,176,209,210,486,508,519,534,535,536,604	Use CustomError Instead of String	Ownable	● Informational	Acknowledged
STEPGO.sol:604	Lack of Error Message	STEPGO	● Informational	Acknowledged
STEPGO.sol:477,534	Cache State Variables That Are Read Multiple Times within a Function	STEPGO	● Informational	Acknowledged
STEPGO.sol:437,438	Variables Can Be Declared as Immutable	STEPGO	● Informational	Acknowledged
STEPGO.sol:496,500	Internal Functions Only Called Once Can Be Inlined	STEPGO	● Informational	Acknowledged
STEPGO.sol:488	Get Contract Balance of ETH in Assembly	STEPGO	● Informational	Acknowledged



Location	Title	Scope	Severity	Status
STEPGO.sol:60,158,159,176,209,210,487,535	Use Assembly to Check Zero Address	Ownable	<div><div></div>Informational</div>	Aknowledged
STEPGO.sol:434	Too Many Digits	STEPGO	<div><div></div>Informational</div>	Aknowledged



Code Security – Unchecked Call Return Value

Title	Severity	Location	Status
Unchecked Call Return Value	● Medium	STEPGO.sol:493	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	STEPGO.sol:106	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	STEPGO.sol:115	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

STEPGO.sol:120

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

STEPGO.sol:138

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

STEPGO.sol:488,493

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.



Code Security – Uninitialized Variables

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

Uninitialized Variables

● Low

STEPGO.sol:442

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

STEPGO.sol:488

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

STEPGO.sol:2,4,37,242,428

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 – Unused Events

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

Unused Events	● Informational	STEPGO.sol:230,243,244,263,264,265,273,451	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	STEPGO.sol:507,518	Aknowledged
--	------------------------------	--------------------	-------------

Description

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

Optimization Suggestion – Variables Should Be Constants

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

Variables Should Be Constants	● Informational	STEPGO.sol:440	Aknowledged
-------------------------------	------------------------------	----------------	-------------

Description

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



Optimization Suggestion – Function Visibility Can Be External

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

Function Visibility Can Be External

● Informational

STEPGO.sol:138,143,5
14,523

Aknowledged

Description

Functions that are not called should be declared as external.

Optimization Suggestion – Floating Pragma

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

Floating Pragma

● Informational

STEPGO.sol:2

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

STEPGO.sol:51,60,127,
145,158,159,164,176,2
09,210,486,508,519,53
4,535,536,604

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

STEPGO.sol:604

Acknowledged

Description

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

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

STEPGO.sol:477,534

Acknowledged

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

STEPGO.sol:437,438

Acknowledged

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 – Internal Functions Only Called Once Can Be Inlined

Title	Severity	Location	Status
Internal Functions Only Called Once Can Be Inlined	● Informational	STEPGO.sol:496,500	Aknowledged

Description

Inlining internal functions that are only called once into the external function can save gas. When compiler optimization is turned off, deploying the contract can save approximately 3000 gas, and calling the function can save approximately 40 gas. When compiler optimization is turned on, deploying the contract can save approximately 2000 gas, and calling the function can save approximately 50 gas.

Optimization Suggestion – Get Contract Balance of ETH in Assembly

Title	Severity	Location	Status
Get Contract Balance of ETH in Assembly	● Informational	STEPGO.sol:488	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	STEPGO.sol:60,158,159,176,209,210,487,535	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	STEPGO.sol:434	Aknowledged

Description

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

The diagram illustrates the relationships between Solidity contracts. The top row contains three contracts: **ERC20** (Context), **UniswapV2Router** (Context), and **UniswapV2Router02** (Context). Below **ERC20** is **Context** (Context). Below **UniswapV2Router** is **UniswapV2Router01** (Context). Below **UniswapV2Router02** is **UniswapV2Router01** (Context). Arrows indicate dependencies: **ERC20** depends on **Context** and **UniswapV2Router01**. **UniswapV2Router** depends on **ERC20**, **UniswapV2Router01**, and **UniswapV2Router02**. **UniswapV2Router01** depends on **UniswapV2Router02**. **Context** depends on **ERC20**. **UniswapV2Router01** depends on **UniswapV2Router02**. **UniswapV2Router02** depends on **UniswapV2Router01**.



Appendix

Finding Categories

Security and Best Practices

- 1.Unchecked Call Return Value: Contracts should handle the return value of external calls to prevent unexpected behavior and vulnerabilities.
- 2.Unauthenticated Storage Access: Smart contracts should undergo scrutiny for unauthenticated storage access, which can lead to unauthorized data tampering.
- 3.Use Safer Functions: Utilize functions known for their secure design to mitigate potential security vulnerabilities. Review functions for enhanced security.
- 4.Uninitialized Variables: Always initialize variables to avoid unexpected behavior and potential vulnerabilities due to uninitialized state.
- 5.Prefer .call() To send()/transfer(): Employ .call() instead of send()/transfer() for external contract calls to minimize security risks associated with reentrancy attacks.
- 6.Recommend to Follow Code Layout Conventions: Strict adherence to established code layout conventions can significantly improve code readability and maintainability.
- 7.Unused Events: Unused events should be removed from contracts to reduce gas consumption and improve contract efficiency.
- 8.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 unexpected behavior.
- 9.Variables Should Be Constants: Declare variables as constants when their values should not change during contract execution to enhance security and readability.
- 10.Function Visibility Can Be External: Enhance gas efficiency by setting functions to external visibility if they are accessible only from within the contract.
- 11.Floating Pragma: Ensure that your Solidity pragma remains consistent for added contract security.
- 12.Use CustomError Instead of String: Opt for custom error codes instead of string error messages for more efficient contract operation and better error handling.
- 13.Lack of Error Message: Include informative error messages in revert/require statements to provide users with clear feedback and facilitate debugging.
- 14.Cache State Variables That Are Read Multiple Times within a Function: Cache frequently accessed state variables within a function to reduce gas costs and improve efficiency.
- 15.Variables Can Be Declared as Immutable: Declare variables as immutable if their values do not change after initialization to enhance security and readability.
- 16.Internal Functions Only Called Once Can Be Inlined: Inlining internal functions that are called only once can optimize gas usage and improve contract efficiency.
- 17.Get Contract Balance of ETH in Assembly: Use assembly to efficiently retrieve the contract's ETH balance and optimize gas usage.
- 18.Use Assembly to Check Zero Address: Optimized assembly checks can be employed to verify zero addresses efficiently.
- 19.Too Many Digits: Avoid excessive precision in numeric calculations to prevent unexpected behavior and 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://stepgo.world/>



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

i 3 Attention

Missing X-Frame-Options Header

YES i

Missing HSTS header

NO ✓

Missing X-Content-Type-Options Header

YES i

Missing Content Security Policy (CSP)

YES i

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

i 3 Attention

Missing SPF Record

NO ✓

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

2 Passed

8 Failed

X (Twitter)



PASS

Facebook

FAIL

Instagram

FAIL

TikTok

FAIL

YouTube

FAIL

Twich

FAIL

Telegram



PASS

Discord

FAIL

Medium

FAIL

Others

FAIL

Recommendation

To enhance project credibility and outreach, we suggest having a minimum of three active social media channels and a fully functional website.

Social Media Information Notes

Unspecified Auditor Notes

Notes from the Project Owner



Fundamental Health

KYC Status

SphinxShield KYC

NO 

3rd Party KYC

NO 

Project Maturity Metrics

Minimally Developed

Low

Token Launch Date

NOT AVAILABLE

Token Market Cap (estimate)

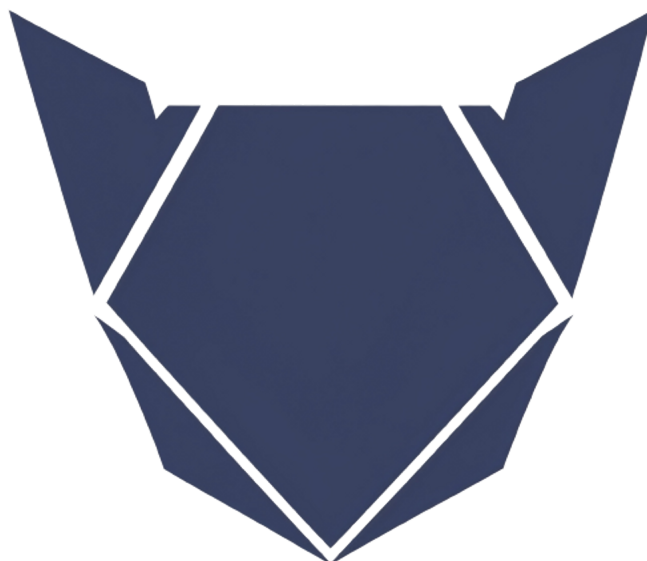
NOT AVAILABLE

Token/Project Age

9 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

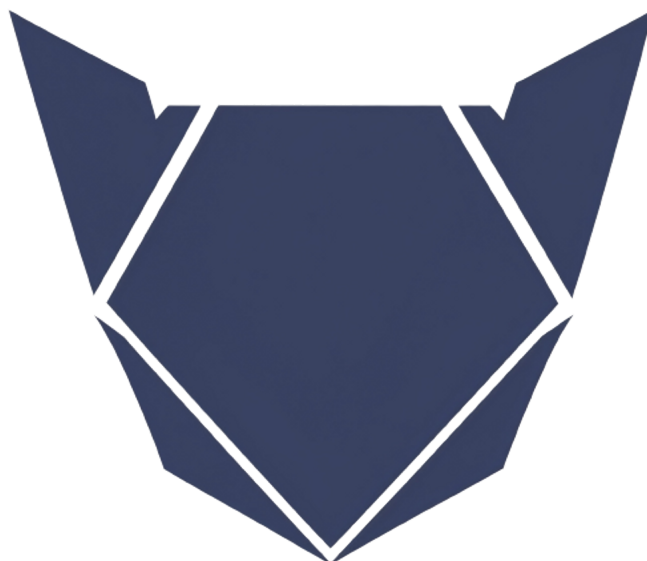
NO 

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.

