

SPHINXSHIELD

Security Assessment

Einsteinereum

Oct 19th, 2023

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	89/100
Auditor Score	86/100

Review by Section	Score
Manual Scan Score	32/57
Advance Check Score	15/19

Scoring System

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

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

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





Table of Contents

Summary

Overview

[Project Summary](#)

[Audit Summary](#)

[Vulnerability Summary](#)

[Audit Scope](#)

Findings

PlantUML

Appendix

Website Scan

Social Media Checks

Fundamental Health

Disclaimer

About



Summary

This audit report is tailored for **Einsteinereum**, aiming to uncover potential issues and vulnerabilities within the **Einsteinereum** 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	Einsteinereum
Blockchain	Ethereum
Language	Solidity
Codebase	https://etherscan.io/address/0x03a371d222ab2b2408dd7689162597e8b68871fc
Commit	

Audit Summary

Delivery Date	Oct 19th, 2023
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Vulnerability Level	Total	⚠ Pending	⊗ Declined	ℹ Acknowledged	✓ Resolved
● High	0	0	0	0	0
● Medium	1	0	0	1	0
● Low	1	0	0	1	0
● Informational	13	0	0	13	0
● Discussion	0	0	0	0	0

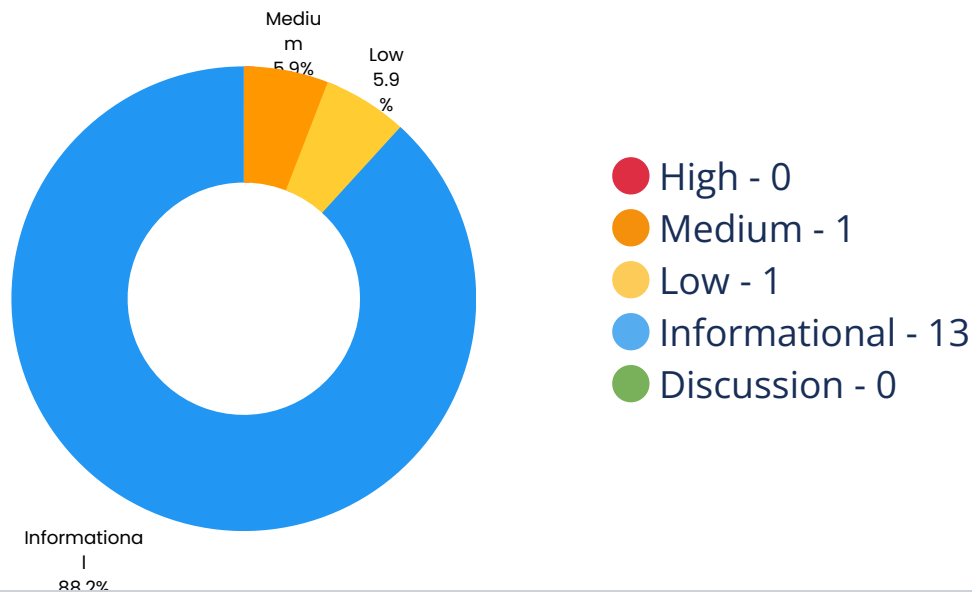


Audit Scope

ID	File	KECCAK256 or SHA256 Checksum
SPM	Einsteinereum .sol	0x6f1b1e5f6c776af3c63c942ec3d3210887255b87dbb485c0cc857b0a4ee549ff



Findings



Location	Title	Scope	Severity	Status
Einsteinereum.sol:462	Unauthenticated Storage Access	StandardToken	Medium	Aknowledged
Einsteinereum.sol:477	Use Safer Functions	StandardToken	Low	Aknowledged
Einsteinereum.sol:477	Prefer .call() To send()/transfer()	StandardToken	Informational	Aknowledged
Einsteinereum.sol:459	Prefer uint256	StandardToken	Informational	Aknowledged
Einsteinereum.sol:452	Set the Constant to Private	StandardToken	Informational	Aknowledged
Einsteinereum.sol:10, 131	Recommend to Follow Code Layout Conventions	IERC20	Informational	Aknowledged
Einsteinereum.sol:462	No Check of Address Params with Zero Address	StandardToken	Informational	Aknowledged
Einsteinereum.sol:449	No Need To Use SafeMath in Solidity Contract of Version 0.8.0 and Above	StandardToken	Informational	Aknowledged



Location	Title	Scope	Severity	Status
Einsteinereum.sol:593,483,540,515,570,491,508,522,553,649,622	Function Visibility Can Be External	StandardToken	● Informational	Acknowledged
Einsteinereum.sol:442	FloatingPragma	Global	● Informational	Acknowledged
Einsteinereum.sol:684,685,758,359,154,174,707,759	Use CustomError Instead of String	Ownable	● Informational	Acknowledged
Einsteinereum.sol:684,685,758,359,174,759	Long String in revert/require	Ownable	● Informational	Acknowledged
Einsteinereum.sol:460,459	Variables Can Be Declared as Immutable	StandardToken	● Informational	Acknowledged
Einsteinereum.sol:790	Empty Function Body	StandardToken	● Informational	Acknowledged
Einsteinereum.sol:684,685,758,174,707,759	Use Assembly to Check Zero Address	Ownable	● Informational	Acknowledged



Code Security – Unauthenticated Storage Access

Title	Severity	Location	Status
Unauthenticated Storage Access	● Medium	Einsteinereum.sol:462	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	Einsteinereum.sol:477	Aknowledged

Description

When calling the transfer, transferFrom, and approve functions in the ERC20 contract, there are some contracts that are not fully implemented in accordance with the ERC20 standard. In order to more comprehensively judge whether the call result meets expectations or to be compatible with different ERC20 contracts, it is recommended to use the safeTransfer, safeTransferFrom, safeApprove function to call.

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

Title	Severity	Location	Status
Prefer .call() To send()/transfer()	● Informational	Einsteinereum.sol:477	Aknowledged

Description

The send or transfer function has a limit of 2300 gas.



Optimization Suggestion – Prefer uint256

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

Prefer uint256

● Informational

Einsteinereum.sol:459

Aknowledged

Description

It is recommended to use uint256/int256 types to avoid gas overhead caused by 32 bytes padding.

Optimization Suggestion – Set the Constant to Private

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

Set the Constant to Private

● Informational

Einsteinereum.sol:452

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 – No Check of Address Params with Zero Address

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

No Check of Address Params with Zero Address

● Informational

Einsteinereum.sol:462

Aknowledged

Description

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



Optimization Suggestion – Recommend to Follow Code Layout Conventions

Title	Severity	Location	Status
Recommend to Follow Code Layout Conventions	● Informational	Einsteinereum.sol:10, 131	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 – No Check of Address Params with Zero Address

Title	Severity	Location	Status
No Check of Address Params with Zero Address	● Informational	Einsteinereum.sol:462	Acknowledged

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	Einsteinereum.sol:449	Acknowledged

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 – Function Visibility Can Be External

Title	Severity	Location	Status
Function Visibility Can Be External	● Informational	Einsteinereum.sol:593 ,483,540,515,570,491, 508,522,553,649,622	Aknowledged

Description

Functions that are not called should be declared as external.

Optimization Suggestion – FloatingPragma

Title	Severity	Location	Status
Floating Pragma	● Informational	Einsteinereum.sol:442	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	Einsteinereum.sol:684 ,685,758,359,154,174, 707,759	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 – Long String in revert/require

Title	Severity	Location	Status
Long String in revert/require	● Informational	Einsteinereum.sol:684,685,758,359,174,759	Acknowledged

Description

If the string parameter in the revert/require function exceeds 32 bytes, more gas will be consumed.

Optimization Suggestion – Variables Can Be Declared as Immutable

Title	Severity	Location	Status
Variables Can Be Declared as Immutable	● Informational	Einsteinereum.sol:460,459	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 – Empty Function Body

Title	Severity	Location	Status
Empty Function Body	● Informational	Einsteinereum.sol:790	Acknowledged

Description

The body of this function is empty.



Optimization Suggestion – Use Assembly to Check Zero Address

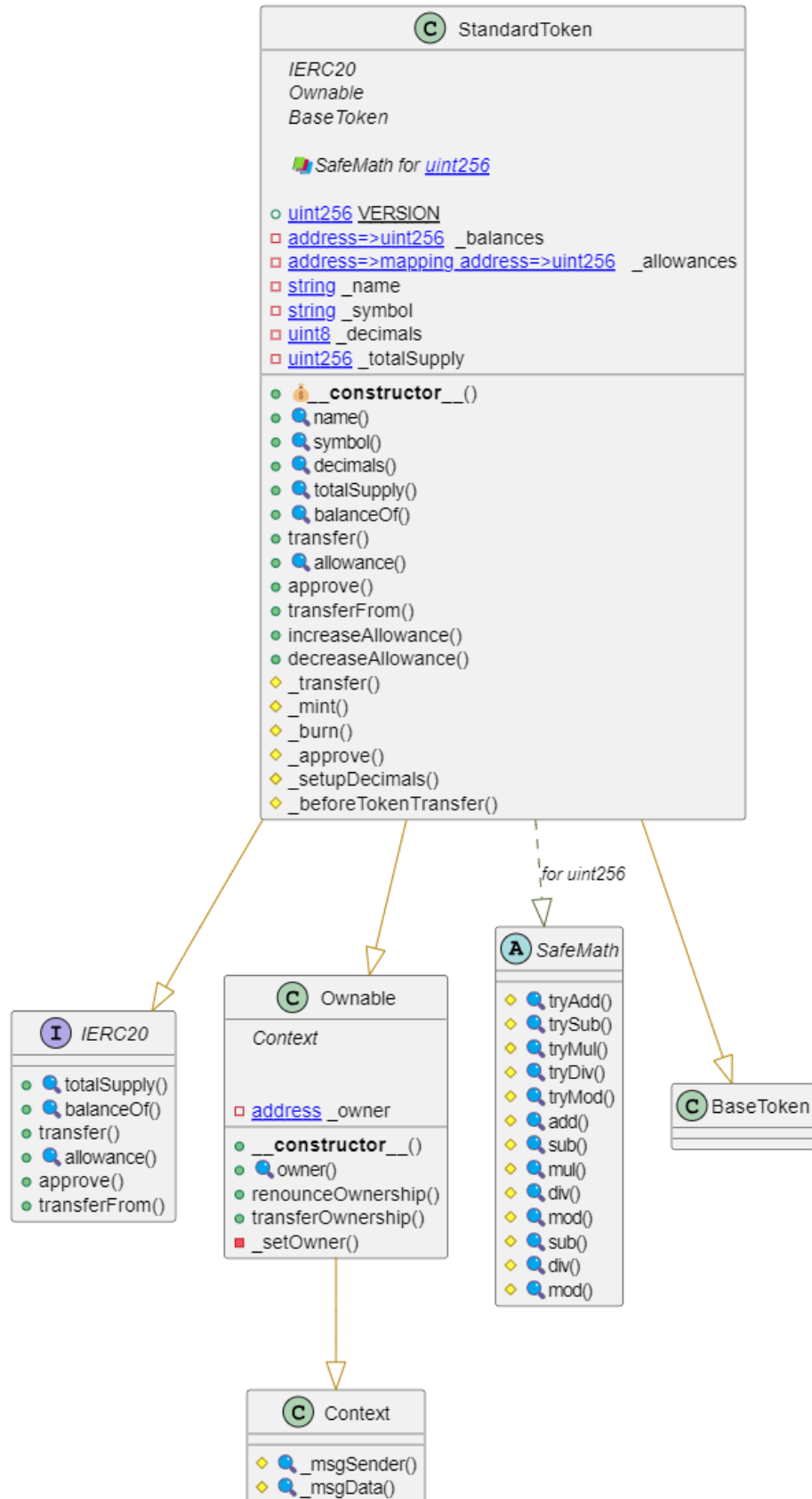
Title	Severity	Location	Status
Use Assembly to Check Zero Address	● Informational	Einsteinereum.sol:684,685,758,174,707,759	Acknowledged

Description

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



PlantUML





Appendix

Finding Categories

Security and Best Practices

- 1.Unauthenticated Storage Access: Smart contracts should undergo scrutiny for unauthenticated storage access, which can lead to unauthorized data tampering.
- 2.Use Safer Functions: Utilize functions known for their secure design to mitigate potential security vulnerabilities. Review functions for enhanced security.
- 3.Prefer .call() To send()/transfer(): Employ .call() instead of send()/transfer() for external contract calls to minimize security risks.
- 4.Prefer uint256: Emphasize the use of uint256 over other data types to maintain consistency and enhance contract security.
- 5.Set the Constant to Private: Declared constants should be set to private visibility to prevent unwanted external access.
- 6.Recommend to Follow Code Layout Conventions: Strict adherence to established code layout conventions can significantly improve code readability and maintainability.
- 7.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.
- 8.No Need To Use SafeMath in Solidity Contract of Version 0.8.0 and Above: Solidity versions 0.8.0 and above feature built-in overflow and underflow protection, minimizing the necessity of SafeMath library usage.
- 9.Function Visibility Can Be External: Enhance gas efficiency by setting functions to external visibility if they are accessible only from within the contract.
- 10.FloatingPragma: Ensure that your Solidity pragma remains consistent for added contract security.
- 11.Use CustomError Instead of String: Opt for custom error codes instead of string error messages for more efficient contract operation.
- 12.Long String in revert/require: Long revert or require strings can increase gas usage and should be optimized for gas efficiency.
- 13.Variables Can Be Declared as Immutable: Variables that do not change after initialization can be declared as immutable to enhance security and readability.
- 14.Empty Function Body: Functions should not contain empty bodies, as this can introduce vulnerabilities.
- 15.Use Assembly to Check Zero Address: Optimized assembly checks can be employed to verify zero addresses efficiently.
- 16.Secure Project Management: Adhering to best practices in project management ensures secure and efficient development processes.
- 17.Code Documentation: Comprehensive code documentation is essential for team collaboration and future code maintenance.
- 18.Trusted Sources for External Contracts: Ensure that external contracts are sourced from reputable and verified developers.
- 19.Regular Code Audits: Regular code audits should be performed to identify and resolve security and functionality issues.



KECCAK256 or SHA256 Checksum Verification

Checksum verification is a critical component of smart contract development. It ensures the integrity of contract deployment and code execution by confirming that the bytecode being executed matches the intended source code. The following details the KECCAK256 and SHA256 checksum verification process.

KECCAK256 Checksum Verification:

- **Checksum Definition:** KECCAK256 is a cryptographic hashing function used in Ethereum to create a checksum of the contract bytecode. It is part of the Ethereum Name Service (ENS) standard.
- **Use Cases:** KECCAK256 checksums are used in ENS for verification of Ethereum addresses. They help prevent unintended transfers due to typos or errors.
- **Checksum Process:** The KECCAK256 checksum is created by taking the SHA3 hash of the lowercase hexadecimal Ethereum address, and then converting it to the corresponding checksum address by replacing characters with uppercase letters.

SHA256 Checksum Verification:

- **Checksum Definition:** SHA256 is a widely used cryptographic hash function, often employed to verify the integrity of data and contracts.
- **Use Cases:** SHA256 checksums are widely used in software development, including the verification of software downloads and smart contracts.
- **Checksum Process:** The SHA256 checksum is generated by applying the SHA256 hashing algorithm to the content of the contract. This results in a fixed-length hexadecimal value that is compared to the expected value to verify the contract's integrity.

Importance of Checksum Verification:

- Checksum verification ensures that smart contracts are executed as intended, preventing tampering and security vulnerabilities.
- It is a security best practice to verify that the deployed bytecode matches the intended source code, reducing the risk of unexpected behavior.

Best Practices:

- Always use checksum verification in situations where it is essential to verify Ethereum addresses or contract integrity.
- Implement checksum verification to ensure that contract deployment and interactions occur as intended.
- Verify the validity of contract deployments and the integrity of the code during development and deployment phases.



Website Scan

 <https://einsteinereum.com/>



Network Security

High | 0 Attentions

Application Security

High | 4 Attentions










DNS Security

High | 0 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

✓ 7 Passed

i 4 Attention

Missing X-Frame-Options Header	YES	i
Missing HSTS header	YES	i
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

✓ 10 Passed

i 0 Attention

Missing SPF Record	NO	✓
Missing DMARC Record	NO	✓
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	NO	✓



Social Media Checks

2 Passed

7 Failed

X (Twitter)



PASS

Facebook

FAIL

Instagram

FAIL

TikTok

FAIL

YouTube

FAIL

Twich

FAIL

Telegram



PASS

Discord

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

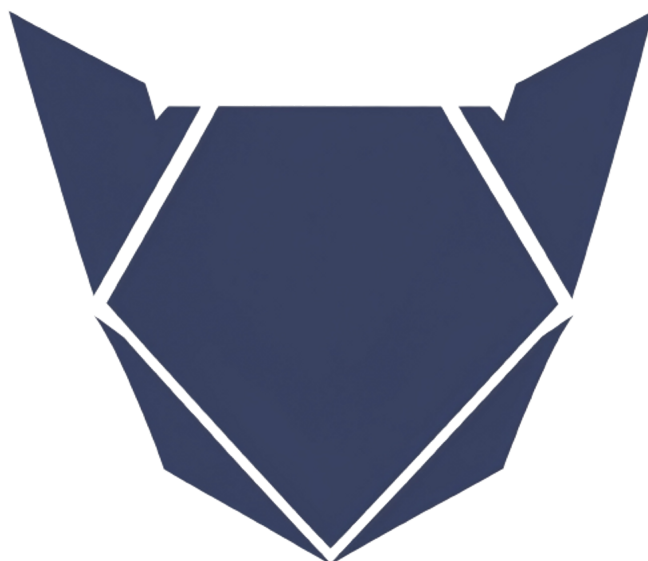
NOT AVAILABLE

Token/Project Age

NOT AVAILABLE

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.





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.

