SPHINXSHIELD

Security Assessment

Liza Token

Nov 9th, 2023

Evaluation Outcomes

Security Score

Review	Score
Overall Score	89/100
Auditor Score	81/100

Review by Section	Score
Manual Scan Score	46/57
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

This audit report is tailored for **Liza Token**, aiming to uncover potential issues and vulnerabilities within the **Liza Token** 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

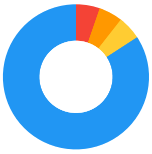
Project Summary

Project Name	Liza Token
Blockchain	Ethereum
Language	Solidity
Codebase	https://etherscan.io/token/0x8b227d72570d3ead66014bca8305cbef7f90d1ee
Commit	5812c58c595b779375ba842971384339950 08e24699e995cd4512a7aeb66aa8b

Audit Summary

Delivery Date	Nov 9th, 2023
Audit Methodology	Static Analysis, Manual Review
Key Components	LizaToken.sol

Vulnerability Summary



19 Total Findings

0	0	0	19	0
Resolved	Mitigated	Partially Resolved	Aknowledged	Declined

Vulnerability Level	Total	Pending	Declined	Aknowledged	Resolved
High	1	0	0	1	0
Medium	1	0	0	1	0
Low	1	0	0	1	0
Informational	16	0	0	16	0
Discussion	0	0	0	0	0

Audit Scope

ID	File	KECCAK256 or SHA256 Checksum
LIZ	LizaToken.sol	0x09b89b050e6415525ccb4660b0305d583504045c569f17a54cbd9ced1355f31f

Understandings

Liza Token is a dynamic deflationary token developed on the Ethereum blockchain. With a burning mechanism, $LIZA offers more than just token utility —it's an AI-powered guide in the crypto landscape. Let's delve into the core aspects of the Liza Token contract:

Token Information

- Token Name: Liza Token
- Symbol: LIZA
- Decimals: 18
- Total Supply: 777,777,777 LIZA

Tax Distribution

Liza Token employs a fee system that divides transactions into various components:

- Dev Fee: Allocated for development purposes. Configurable by the owner.
- Marketing Fee: Reserved for marketing efforts. Owner-adjustable.
- Rewards Fee: A portion goes to reward mechanisms. Owner-adjustable.
- Burn Fee: Tokens are burned, reducing the total supply. Adjustable by the owner.
- Total Fee: Sum of the above fees.
- Fee Denominator: Denominator used in fee calculations, typically set to 100.

Fee Management

The contract allows the owner to manage fees, adjusting liquidity, team, marketing, dev, and burn fees, along with fee multipliers and receivers.

Tax Exemption

Owners can exempt specific addresses from fees, providing flexibility and whitelisting options.

Ownership and Authorization

The contract owner can authorize specific addresses with privileged functions, enhancing security and configuration capabilities.

Transaction Limits

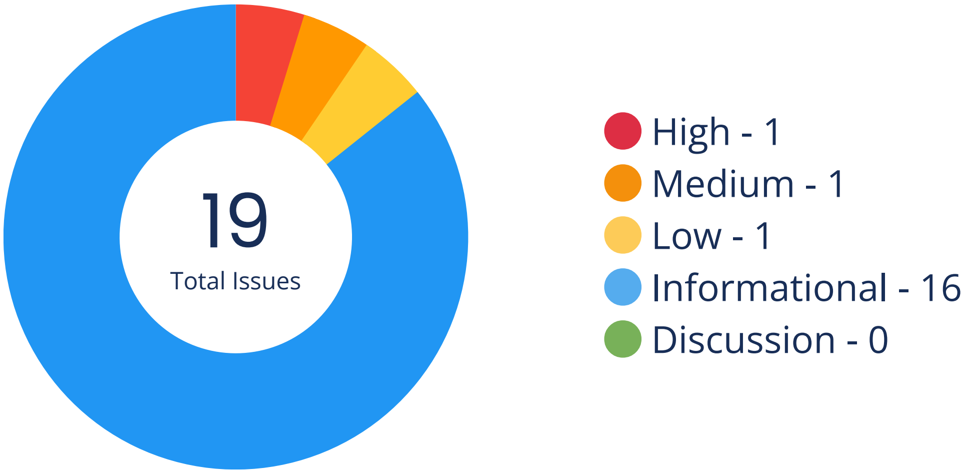
To prevent excessive token movement, the contract enforces transaction limits. Swap Mechanism: Liza Token utilizes a swap mechanism for managing liquidity. A portion of the contract's balance is swapped to BB tokens via PancakeSwap when a set threshold is reached.

Open Trading

Trading can be restricted based on owner-defined conditions, ensuring controlled and secure trading environments.
Additional Functionality: The contract includes various functions, such as clearing stuck ETH and tokens, enhancing its utility.

This overview provides insights into the robust features and functions of the Liza Token contract on the Ethereum blockchain. Please note that the contract plays a pivotal role in governing the project's operational aspects, including fees, liquidity, and user interactions.

Findings



Location	Title	Scope	Severity	Status
LizaToken.sol:888	Reentrancy	LizaToken	High	Aknowledged
LizaToken.sol:847,848,849,899	Unchecked Call Return Value	LizaToken	Medium	Aknowledged
LizaToken.sol:664,665,666	Uninitialized Variables	LizaToken	Low	Aknowledged
LizaToken.sol:3,44,633	Recommend to Follow Code Layout Conventions	Ownable	Informational	Aknowledged
LizaToken.sol:30,114,195,224,588	Specify Multiple Compiler Versions	Global	Informational	Aknowledged
LizaToken.sol:855,902,961	No Check of Address Params with Zero Address	LizaToken	Informational	Aknowledged
LizaToken.sol:932,938	Use Shift Operation Instead of Mul/Div	LizaToken	Informational	Aknowledged
LizaToken.sol:474,477,863,890,903	Continuous State Variable Write	LizaToken	Informational	Aknowledged

Location	Title	Scope	Severity	Status
LizaToken.sol:695,723,724,725,726,727,823,829,833,971	Cache State Variables that are Read Multiple Times within A Function	LizaToken	Informational	Aknowledged
LizaToken.sol:823	Use != 0 Instead of > 0 for Unsigned Integer Comparison	LizaToken	Informational	Aknowledged
LizaToken.sol:280,288,305,331,354,376,395,415,902	Function Visibility Can Be External	ERC20	Informational	Aknowledged
LizaToken.sol:3,30,114,195,224	Floating Pragma	Global	Informational	Aknowledged
LizaToken.sol:75,94,418,441,442,447,470,496,501,527,528,545,857,889,908,909,910,920,921,926,932,938,949,958,962,983	Use CustomError Instead of String	Ownable	Informational	Aknowledged
LizaToken.sol:94,418,441,442,447,496,501,527,528,857,889,920,921,926,932,938,962,983	Long String in revert/require	Ownable	Informational	Aknowledged
LizaToken.sol:637,638,644,646	Variables Can Be Declared as Immutable	LizaToken	Informational	Aknowledged
LizaToken.sol:841	Get Contract Balance of ETH in Assembly	LizaToken	Informational	Aknowledged
LizaToken.sol:566,582	Empty Function Body	ERC20	Informational	Aknowledged
LizaToken.sol:94,441,442,470,496,527,528,908,909,910	Use Assembly to Check Zero Address	Ownable	Informational	Aknowledged
LizaToken.sol:920	Too Many Digits	LizaToken	Informational	Aknowledged

Code Security - Reentrancy

Title	Severity	Location	Status
Reentrancy	High	LizaToken.sol:888	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	LizaToken.sol:847,848, 849,899	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 - Uninitialized Variables

Title	Severity	Location	Status
Uninitialized Variables	Low	LizaToken.sol:664,665, 666	Aknowledged

Description

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

Optimization Suggestion - Recommend to Follow Code Layout Conventions

Title	Severity	Location	Status
Recommend to Follow Code Layout Conventions	Informational	LizaToken.sol:3,44,633	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 - Specify Multiple Compiler Versions

Title	Severity	Location	Status
Specify Multiple Compiler Versions	Informational	LizaToken.sol:30,114,195,224,588	Aknowledged

Description

Multiple compiler versions are specified in one contract file.

Optimization Suggestion - No Check of Address Params with Zero Address

Title	Severity	Location	Status
No Check of Address Params with Zero Address	Informational	LizaToken.sol:855,902,961	Aknowledged

Description

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

Optimization Suggestion - Use Shift Operation Instead of Mul/Div

Title	Severity	Location	Status
Use Shift Operation Instead of Mul/Div	Informational	LizaToken.sol:932,938	Aknowledged

Description

It is recommended to use shift operation instead of direct multiplication and division if possible, because shift operation is more gas-efficient.

Optimization Suggestion - Continuous State Variable Write

Title	Severity	Location	Status
Continuous State Variable Write	Informational	LizaToken.sol:474,477, 863,890,903	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 - 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	LizaToken.sol:695,723, 724,725,726,727,823,8 29,833,971	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 - Use != 0 Instead of > 0 for Unsigned Integer Comparison

Title	Severity	Location	Status
Use != 0 Instead of > 0 for Unsigned Integer Comparison	Informational	LizaToken.sol:823	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	LizaToken.sol:280,288,305,331,354,376,395,415,902	Aknowledged

Description

Functions that are not called should be declared as external.

Optimization Suggestion - Floating Pragma

Title	Severity	Location	Status
Floating Pragma	Informational	LizaToken.sol:3,30,114,195,224	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	LizaToken.sol:75,94,418,441,442,447,470,496,501,527,528,545,857,889,908,909,910,920,921,926,932,938,949,958,962,983	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	LizaToken.sol:94,418,441,442,447,496,501,527,528,857,889,920,921,926,932,938,962,983	Aknowledged

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	LizaToken.sol:637,638,644,646	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	LizaToken.sol:841	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 - Empty Function Body

Title	Severity	Location	Status
Empty Function Body	Informational	LizaToken.sol:566,582	Aknowledged

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	LizaToken.sol:94,441,442,470,496,527,528,908,909,910	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	LizaToken.sol:920	Aknowledged

Description

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

PlantUML


LizaToken
ERC20
Ownable
IUniswapV2Factory
createPair()
IUniswapV2Router
ERC20
Context
IERC20
IERC20Metadata
Ownable
Context
IERC20Metadata
IERC20

Appendix

Finding Categories

Security and Best Practices

1. Reentrancy: Exercise caution to prevent reentrancy attacks by carefully managing state changes and using mutex patterns.
2. Unchecked Call Return Value: Always check and handle the return values of external calls to avoid unexpected behavior.
3. Uninitialized Variables: Ensure that all variables are properly initialized to prevent unpredictable outcomes and vulnerabilities.
4. Recommend to Follow Code Layout Conventions: Adhering to established code layout conventions enhances readability and maintainability.
5. Specify Multiple Compiler Versions: Clearly specify the Solidity compiler version to avoid compatibility issues and ensure code consistency.
6. No Check of Address Params with Zero Address: Always verify address parameters to ensure they are not the zero address, preventing unexpected behavior.
7. Use Shift Operation Instead of Mul/Div: Opt for shift operations over multiplication/division for gas-efficient arithmetic operations.
8. Continuous State Variable Write: Minimize state variable writes within a function to reduce gas costs and enhance efficiency.
9. Cache State Variables that are Read Multiple Times within A Function: Improve efficiency by caching state variables that are read multiple times in a function.
10. Use != 0 Instead of > 0 for Unsigned Integer Comparison: Prefer using != 0 for clarity in unsigned integer comparisons, enhancing code readability.
11. Function Visibility Can Be External: Set external visibility for functions only accessible within the contract to enhance gas efficiency.
12. Floating Pragma: Maintain a consistent Solidity pragma version for added contract security and stability.
13. Use CustomError Instead of String: Employ custom error codes instead of string error messages for more efficient contract operation.
14. Long String in revert/require: Optimize gas efficiency by minimizing the length of strings used in revert/require statements.
15. Variables Can Be Declared as Immutable: Declare variables as immutable if they do not change after initialization to enhance security and readability.
16. Get Contract Balance of ETH in Assembly: Use optimized assembly to fetch the contract's ETH balance efficiently.
17. Empty Function Body: Avoid empty function bodies, as they can introduce vulnerabilities and lead to unexpected behavior.
18. Use Assembly to Check Zero Address: Employ assembly checks for efficient verification of zero addresses.
19. Too Many Digits: Be cautious of using an excessive number of digits, as it can impact gas costs and contract efficiency.

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://lizatoken.com/

Network Security

High | 0 Attentions

Application Security

High | 1 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

10 Passed | 1 Attention

Missing X-Frame-Options Header	NO
Missing HSTS header	NO
Missing X-Content-Type-Options Header	NO
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	YES
Missing DKIM Record	NO
Ineffective SPF Record	YES
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

5 Passed 5 Failed

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

Recommendation

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

Social Media Information Notes

Unspecified Auditor Notes

Notes from the Project Owner

Fundamental Health

KYC Status

SphinxShield KYC	NO
3rd Party KYC	NO

Project Maturity Metrics

Somewhat Developed	**MEDIUM**
Token Launch Date	**2023.07.09 10:30 (UTC)**
Token Market Cap (estimate)	**$396.50K**
Token/Project Age	**122 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	NO
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

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.