SPHINXSHIELD

Security Assessment

Troll

Feb 13th, 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	82/100
Auditor Score	80/100

Review by Section	Score
Manual Scan Score	51/57
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.

Audit Passed

PASSED

Table of Contents

Summary

This audit report is tailored for **Troll**, aiming to uncover potential issues and vulnerabilities within the **Troll** 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	Troll
Blockchain	Ethereum
Language	Solidity
Codebase	https://etherscan.io/token/0xf8ebf4849f1fa4faf0dff2106a173d3a6cb2eb3a
Commit	f40957fe9d202c048636b57f8c86ff96b28819843125c88ae65585282b772e5a

Audit Summary

Delivery Date	Feb 13th, 2024
Audit Methodology	Static Analysis, Manual Review
Key Components	TrollFace.sol

Vulnerability Summary

19	0	0	0	19	0
Total Findings	Resolved	Mitigated	Partially Resolved	Aknowledged	Declined

Vulnerability Level	Total	Pending	Declined	Aknowledged	Resolved
● High	1	0	0	1	0
● Medium	4	0	0	4	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
TLF	TrollFace.sol	0xa0ee45b5539fcaaccbed16d94f27ce355f280bf08381fa75c2ecda277834cb4c

Understandings

Troll Coin ($TROLL) is a memebreaking cryptocurrency embracing internet culture. Deployed on the Ethereum network, the TROLL contract is an ERC20 token with various features and mechanisms designed to enhance user experience and facilitate community engagement.

Token Information

- Token Name: TROLL
- Symbol: TROLL
- Decimals: 18
- Total Supply: 960,420,000,000 TROLL

Fee Management

- Troll implements fees for transactions, including marketing, liquidity, and total fees.
- Owners can adjust fee percentages and denominators to optimize fee structure.
- The contract supports different fee settings for buy and sell transactions.

Ownership and Authorization

- The contract owner can authorize specific addresses, granting access to privileged functions.
- These functions are restricted by the onlyOwner modifier, ensuring secure configuration.

Transaction Limits

Transaction limits are enforced to prevent excessive token movement, enhancing stability and security.

Swap Mechanism

- TROLL employs a swap mechanism to manage liquidity, utilizing Uniswap Router for token swaps.
- When a set threshold is reached, tokens are swapped for ETH, ensuring liquidity and price stability.

Open Trading

Trading can be restricted based on owner-defined conditions, ensuring controlled and regulated market activity.

Additional Functionality

The contract includes functions for clearing stuck ETH, clearing tokens, and more, enhancing versatility and usability.

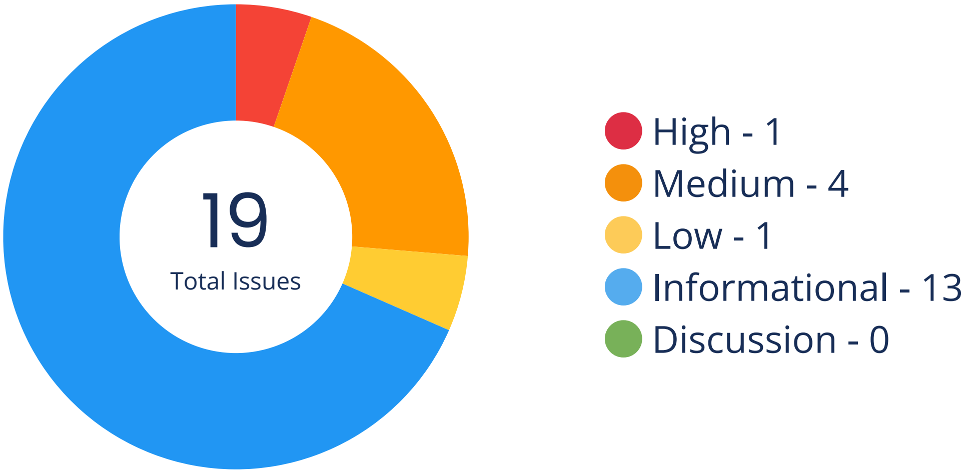
About Troll

TROLL coin embodies the spirit of internet culture, aiming to bring laughter and fun to the cryptocurrency space. With a strong community and a renounced contract, Troll seeks to establish itself as a leading meme coin on the internet.

Operation Troll

The TROLL contract, named Trollface ($TROLL), is an ERC20 token deployed on the Ethereum network. It incorporates various features such as SafeMath for secure arithmetic operations and UniswapV2Router02 for decentralized trading. TROLL tokenomics include customizable fee structures, trading restrictions, and liquidity management mechanisms, ensuring a vibrant and sustainable ecosystem.

Findings



Location	Title	Scope	Severity	Status
TrollFace.sol:602	Freeze Money	TROLLFACE	● High	Aknowledged
TrollFace.sol:111	Unauthenticated Storage Access	ERC20	● Medium	Aknowledged
TrollFace.sol:120	Unauthenticated Storage Access	ERC20	● Medium	Aknowledged
TrollFace.sol:125	Unauthenticated Storage Access	ERC20	● Medium	Aknowledged
TrollFace.sol:141	Unauthenticated Storage Access	ERC20	● Medium	Aknowledged
TrollFace.sol:508,527,528	Uninitialized Variables	TROLLFACE	● Low	Aknowledged
TrollFace.sol:506	Set the Constant to Private	TROLLFACE	● Informational	Aknowledged
TrollFace.sol:3,16,50,339,501	Recommend to Follow Code Layout Conventions	Ownable	● Informational	Aknowledged

Location	Title	Scope	Severity	Status
TrollFace.sol:310,340,345,388,390,398,535,544,549	Unused Events	IUniswapV2Factory	● Informational	Aknowledged
TrollFace.sol:639,667,672	No Check of Address Params with Zero Address	TROLLFACE	● Informational	Aknowledged
TrollFace.sol:3,4	Inconsistent Solidity Version	Global	● Informational	Aknowledged
TrollFace.sol:643,668	Continuous State Variable Write	TROLLFACE	● Informational	Aknowledged
TrollFace.sol:141,146,639,667,672,690	Function Visibility Can Be External	ERC20	● Informational	Aknowledged
TrollFace.sol:3	Floating Pragma	Global	● Informational	Aknowledged
TrollFace.sol:30,39,133,148,161,162,167,179,195,196,614,618,627,631,653,664,676	Use CustomError Instead of String	Ownable	● Informational	Aknowledged
TrollFace.sol:565,663	Cache State Variables That Are Read Multiple Times within a Function	TROLLFACE	● Informational	Aknowledged
TrollFace.sol:510	Variables Can Be Declared as Immutable	TROLLFACE	● Informational	Aknowledged
TrollFace.sol:39,161,162,179,195,196	Use Assembly to Check Zero Address	Ownable	● Informational	Aknowledged
TrollFace.sol:574,576,577,615	Too Many Digits	TROLLFACE	● Informational	Aknowledged

Code Security - Freeze Money

Title	Severity	Location	Status
Freeze Money	● High	TrollFace.sol:602	Aknowledged

Description

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

Code Security - Unauthenticated Storage Access

Title	Severity	Location	Status
Unauthenticated Storage Access	● Medium	TrollFace.sol:111	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	TrollFace.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	TrollFace.sol:125	Aknowledged

Description

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

Code Security - Unauthenticated Storage Access

Title	Severity	Location	Status
Unauthenticated Storage Access	● Medium	TrollFace.sol:141	Aknowledged

Description

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

Code Security - Uninitialized Variables

Title	Severity	Location	Status
Uninitialized Variables	● Low	TrollFace.sol:508,527, 528	Aknowledged

Description

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

Optimization Suggestion - Set the Constant to Private

Title	Severity	Location	Status
Set the Constant to Private	● Informational	TrollFace.sol:506	Aknowledged

Description

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

Optimization Suggestion - Recommend to Follow Code Layout Conventions

Title	Severity	Location	Status
Recommend to Follow Code Layout Conventions	● Informational	TrollFace.sol:3,16,50,339,501	Aknowledged

Description

In the solidity document (https://docs.soliditylang.org/en/v0.8.17/style-guide.html), there are the following conventions for code layout: Layout contract elements in the following order: 1. Pragma statements, 2. Import statements, 3. Interfaces, 4. Libraries, 5. Contracts. Inside each contract, library or interface, use the following order: 1. Type declarations, 2. State variables, 3. Events, 4. Modifiers, 5. Functions. Functions should be grouped according to their visibility and ordered: 1. constructor, 2. receive function (if exists), 3. fallback function (if exists), 4. external, 5. public, 6. internal, 7. private.

Optimization Suggestion - Unused Events

Title	Severity	Location	Status
Unused Events	● Informational	TrollFace.sol:310,340, 345,388,390,398,535,544,549	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	TrollFace.sol:639,667, 672	Aknowledged

Description

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

Optimization Suggestion - Inconsistent Solidity Version

Title	Severity	Location	Status
Inconsistent Solidity Version	● Informational	TrollFace.sol:3,4	Aknowledged

Description

The source files have different solidity compiler ranges referenced. This leads to potential security flaws between deployed contracts depending on the compiler version chosen for any particular file. It also increases the cost of maintenance as different compiler versions have different semantics and behavior.

Optimization Suggestion - Continuous State Variable Write

Title	Severity	Location	Status
Continuous State Variable Write	● Informational	TrollFace.sol:643,668	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 - Function Visibility Can Be External

Title	Severity	Location	Status
Function Visibility Can Be External	● Informational	TrollFace.sol:141,146, 639,667,672,690	Aknowledged

Description

Functions that are not called should be declared as external.

Optimization Suggestion - Floating Pragma

Title	Severity	Location	Status
Floating Pragma	● Informational	TrollFace.sol:3	Aknowledged

Description

Contracts should be deployed with fixed compiler version which has been tested thoroughly or make sure to lock the contract compiler version in the project configuration. Locked compiler version ensures that contracts will not be compiled by untested compiler version.

Optimization Suggestion - Use CustomError Instead of String

Title	Severity	Location	Status
Use CustomError Instead of String	● Informational	TrollFace.sol:30,39,133,148,161,162,167,179,195,196,614,618,627,631,653,664,676	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 - 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	TrollFace.sol:565,663	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	TrollFace.sol:510	Aknowledged

Description

The solidity compiler of version 0.6.5 introduces immutable to modify state variables that are only modified in the constructor. Using immutable can save gas.

Optimization Suggestion - Use Assembly to Check Zero Address

Title	Severity	Location	Status
Use Assembly to Check Zero Address	● Informational	TrollFace.sol:39,161,162,179,195,196	Aknowledged

Description

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

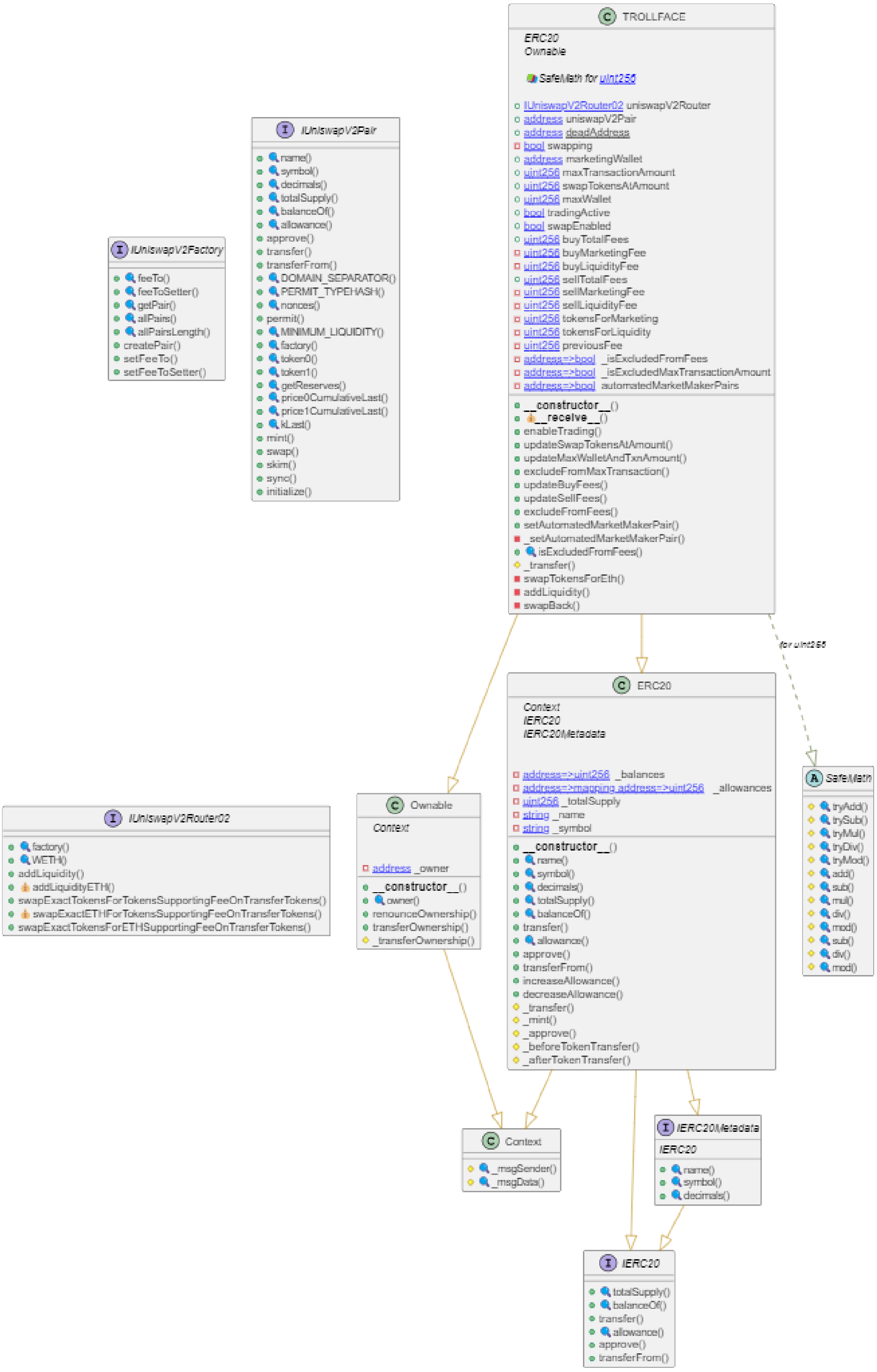
Optimization Suggestion - Too Many Digits

Title	Severity	Location	Status
Too Many Digits	● Informational	TrollFace.sol:574,576,577,615	Aknowledged

Description

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

PlantUML


TROLLFACE
ERC20
Ownable
SafeMath for uint256
IUniswapV2Router02 uniswapV2Router
address uniswapV2Pair
address deadAddress
bool swapping
address marketingWallet
uint256 maxTransactionAmount
uint256 swapTokensAtAmount
uint256 maxWallet
bool tradingActive
bool swapEnabled
uint256 buyTotalFees
uint256 buyMarketingFee
uint256 buyLiquidityFee
uint256 sellTotalFees
uint256 sellMarketingFee
uint256 sellLiquidityFee
uint256 tokensForMarketing
uint256 tokensForLiquidity
uint256 previousFee
address=>bool _isExcludedFromFees
address=>bool _isExcludedMaxTransactionAmount
address=>bool automatedMarketMakerPairs
__constructor__()
__receive__()
enableTrading()
updateSwapTokensAtAmount()
updateMaxWalletAndTxnAmount()
excludeFromMaxTransaction()
updateBuyFees()
updateSellFees()
excludeFromFees()
setAutomatedMarketMakerPair()
_setAutomatedMarketMakerPair()
isExcludedFromFees()
_transfer()
swapTokensForEth()
addLiquidity()
swapBack()
IUniswapV2Pair
name()
symbol()
decimals()
totalSupply()
balanceOf()
allowance()
approve()
transfer()
transferFrom()
DOMAIN_SEPARATOR()
PERMIT_TYPEHASH()
nonces()
permit()
MINIMUM_LIQUIDITY()
factory()
token0()
token1()
getReserves()
price0CumulativeLast()
price1CumulativeLast()
kLast()
mint()
swap()
skim()
sync()
initialize()
IUniswapV2Factory
feeTo()
feeToSetter()
getPair()
allPairs()
allPairsLength()
createPair()
setFeeTo()
setFeeToSetter()
for uint256
ERC20
Context
IERC20
IERC20Metadata
address=>uint256 _balances
address=>mapping address=>uint256 _allowances
uint256 _totalSupply
string _name
string _symbol
__constructor__()
name()
symbol()
decimals()
totalSupply()
balanceOf()
transfer()
allowance()
approve()
transferFrom()
increaseAllowance()
decreaseAllowance()
_transfer()
_mint()
_approve()
_beforeTokenTransfer()
_afterTokenTransfer()
SafeMath
tryAdd()
trySub()
tryMul()
tryDiv()
tryMod()
add()
sub()
mul()
div()
mod()
sub()
div()
mod()
IUniswapV2Router02
factory()
WETH()
addLiquidity()
addLiquidityETH()
swapExactTokensForTokensSupportingFeeOnTransferTokens()
swapExactETHForTokensSupportingFeeOnTransferTokens()
swapExactTokensForETHSupportingFeeOnTransferTokens()
Ownable
Context
address _owner
__constructor__()
owner()
renounceOwnership()
transferOwnership()
_transferOwnership()
Context
_msgSender()
_msgData()
IERC20Metadata
IERC20
name()
symbol()
decimals()
IERC20
totalSupply()
balanceOf()
transfer()
allowance()
approve()
transferFrom()

Appendix

Finding Categories

Security and Best Practices

1. Freeze Money: Implement mechanisms to freeze funds in case of emergencies or suspicious activities to prevent unauthorized withdrawals.
2. Unauthenticated Storage Access: Smart contracts should undergo scrutiny for unauthenticated storage access, which can lead to unauthorized data tampering.
3. Uninitialized Variables: Ensure all variables are properly initialized to prevent unexpected behavior and vulnerabilities.
4. Set the Constant to Private: Declared constants should be set to private visibility to prevent unwanted external access and modifications.
5. Recommend to Follow Code Layout Conventions: Adhere to established code layout conventions to enhance code readability and maintainability.
6. Unused Events: Remove unused event declarations to declutter the contract code and improve code clarity.
7. No Check of Address Params with Zero Address: Verify address parameters to ensure they are not set to the zero address, which could lead to unexpected behavior.
8. Inconsistent Solidity Version: Ensure consistent use of Solidity version across the contract codebase to prevent compatibility issues and unexpected behavior.
9. Continuous State Variable Write: Minimize continuous state variable writes to optimize gas usage and improve contract efficiency.
10. Function Visibility Can Be External: Set function visibility to external if they are only accessible from outside the contract, enhancing gas efficiency.
11. Floating Pragma: Maintain a consistent Solidity pragma directive throughout the contract codebase 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 reduced gas usage.
13. Cache State Variables That Are Read Multiple Times within a Function: Cache state variables that are read multiple times within a function to reduce gas costs and improve performance.
14. Variables Can Be Declared as Immutable: Declare variables as immutable if they do not change after initialization to enhance security and readability.
15. Use Assembly to Check Zero Address: Employ optimized assembly checks to verify zero addresses efficiently and securely.
16. Too Many Digits: Avoid excessive precision in numerical values to prevent potential overflow or underflow issues and ensure contract stability.

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://troll.run/

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

ⓘ 3 Attention

Missing SPF Record	YES	ⓘ
Missing DMARC Record	NO	✓
Missing DKIM Record	NO	✓
Ineffective SPF Record	YES	ⓘ
SPF Record Contains a Softfail Without DMARC	YES	ⓘ
Name Servers Versions Exposed	NO	✓
Allow Recursive Queries	NO	✓
CNAME in NS Records	NO	✓
MX Records IPs are Private	NO	✓
MX Records has Invalid Chars	NO	✓

Social Media Checks

✓ 3 Passed   ⓘ 7 Failed

X (Twitter)	↗	PASS ✓
Facebook		FAIL ✕
Instagram	↗	PASS ✓
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

Moderately Developed	**Medium**
Token Launch Date	**2023.04.19 9:30 (UTC)**
Token Market Cap (estimate)	**$26.03M**
Token/Project Age	**300 Days**

Recommendation

We strongly recommend that the project undergo the Know Your Customer (KYC) verification process with SphinxShield to enhance transparency and build trust within the crypto community. Furthermore, we encourage the project team to reach out to us promptly to rectify any inaccuracies or discrepancies in the provided information to ensure the accuracy and reliability of their project data.

Coin Tracker Analytics

Status

CoinMarketCap	YES	✓
CoinGecko	YES	✓
Others	YES	✓

Recommendation

We highly recommend that the project consider integrating with multiple coin tracking platforms to expand its visibility within the cryptocurrency ecosystem. In particular, joining prominent platforms such as CoinMarketCap and CoinGecko can significantly benefit the project by increasing its reach and credibility.

CEX Holding Analytics

Status

The coin is available on Huobi, Gate.io, MEXC, CoinW, Bitmart, DigiFinex and more...

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.