SPHINXSHIELD

Security Assessment

Oggy Inu

Oct 28th, 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	83/100

Review by Section	Score
Manual Scan Score	38/57
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 **Oggy Inu**, aiming to uncover potential issues and vulnerabilities within the **Oggy Inu** 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

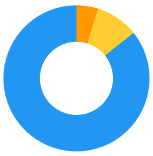
Project Summary

Project Name	Oggy Inu
Blockchain	Ethereum
Language	Solidity
Codebase	https://etherscan.io/token/0x7e877b99897D514da01bD1d177E693EC639961Af
Commit	9f11778286de4fe8bb61e7399e2c25ceb3b27d719ef6998ac91dc4c38c5f0b97

Audit Summary

Delivery Date	Oct 28th, 2023
Audit Methodology	Static Analysis, Manual Review
Key Components	OGGYINU.sol

Vulnerability Summary



23	0	0	0	23	0
Total Findings	Resolved	Mitigated	Partially Resolved	Aknowledged	Declined

Vulnerability Level	Total	(!) Pending	⊗ Declined	ⓘ Aknowledged	✓ Resolved
● High	0	0	0	0	0
● Medium	4	0	0	4	0
● Low	1	0	0	1	0
● Informational	18	0	0	18	0
● Discussion	0	0	0	0	0

Audit Scope

ID	File	KECCAK256 or SHA256 Checksum
BRT	OGGYINU.sol	0x57fedbd9785ae221a53b5e4d7d8ef2150f48a2c7b155415939657d6213d64678

Understandings

BOggy Inu is a decentralized finance (DeFi) token deployed on the Ethereum blockchain. The contract employs various functions and mechanisms to manage its operations, including tax distribution, liquidity acquisition, and privileged functions for modifying contract parameters. Here's a breakdown of key components and functionalities within the contract:

Token Information

- Token Name: Oggy Inu
- Symbol: OGGY
- Decimals: 9
- Total Supply: 420,000,000,000 OGGY

Tax Distribution

Oggy Inu transactions incur a total fee, which is divided into various components:

1. Liquidity Fee: This fee is collected for providing liquidity to the token. Its value is set by the owner.
2. Team Fee: A portion of the fee goes to the project's team. The team fee is adjustable by the owner.
3. Marketing Fee: Part of the fee is allocated to marketing efforts. The marketing fee is configurable by the owner.
4. Dev Fee: A portion of the fee is allocated for development purposes, and its value can be set by the owner.
5. Burn Fee: Some tokens are burned, effectively reducing the total supply. The burn fee can be adjusted by the owner.
6. Total Fee: The sum of the above-mentioned fees forms the total fee collected in each transaction. It is used to calculate the overall fee distribution.
7. Fee Denominator: The denominator used in fee calculations, usually set to 100.

Tax Distribution

Oggy Inu transactions incur a total fee, which is divided into various components:

1. Liquidity Fee: This fee is collected for providing liquidity to the token. Its value is set by the owner.
2. Team Fee: A portion of the fee goes to the project's team. The team fee is adjustable by the owner.
3. Marketing Fee: Part of the fee is allocated to marketing efforts. The marketing fee is configurable by the owner.
4. Dev Fee: A portion of the fee is allocated for development purposes, and its value can be set by the owner.
5. Burn Fee: Some tokens are burned, effectively reducing the total supply. The burn fee can be adjusted by the owner.
6. Total Fee: The sum of the above-mentioned fees forms the total fee collected in each transaction. It is used to calculate the overall fee distribution.
7. Fee Denominator: The denominator used in fee calculations, usually set to 100.

Fee Management

The contract allows the owner to manage various fees:

- Set Tax: The owner can configure the liquidity fee, team fee, marketing fee, dev fee, and burn fee, as well as the fee denominator. These adjustments enable flexibility in managing the fee structure.
- Set Fee Multipliers: The owner can set multipliers to adjust the percentage of fees for buy, sell, and transfer transactions.
- Set Fee Receivers: The addresses that receive the various fee components (auto-liquidity, marketing, dev, and team fees) can be set by the owner.

Tax Exemption

The contract allows the owner to exempt specific addresses from fees, providing a mechanism for fee exemption to certain addresses. This can be used for whitelisting specific wallets or contracts.

Ownership and Authorization

The contract owner can authorize specific addresses, allowing them to access privileged functions. These functions are restricted by the onlyOwner modifier and are used for configuring the contract and address attributes..

Ownership and Authorization

The contract owner can authorize specific addresses, allowing them to access privileged functions. These functions are restricted by the onlyOwner modifier and are used for configuring the contract and address attributes.

Transaction Limits

The contract enforces transaction limits to prevent excessive token movement. It ensures that users do not make transactions that exceed the defined limits.

Swap Mechanism

Oggy Inu employs a swap mechanism that is used to manage liquidity. When a set threshold of tokens is reached, a portion of the contract's balance is swapped to BB tokens via the PancakeSwap Router. This swap action temporarily affects the token's price. The remaining balance is then supplied to the Oggy Inu-BNB liquidity pool.

Open Trading

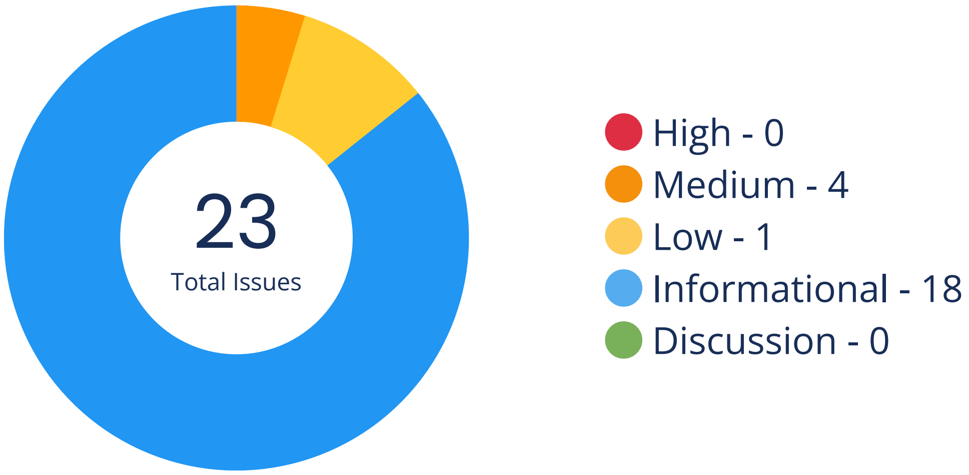
Trading can be restricted based on conditions defined by the owner. This ensures that trading remains closed until specific requirements are met.

Additional Functionality

The contract includes other functions, such as clearing stuck ETH, clearing tokens, and more.

This understanding provides insights into the key features and functions of the Oggy Inu contract deployed on the Ethereum blockchain. Please note that the contract is a vital component of the Oggy Inu project's infrastructure, governing various aspects of its operation, including fees, liquidity, and user interactions.

Findings



Location	Title	Scope	Severity	Status
OGGYINU.sol:385,428,430	Unchecked Call Return Value	OGGYINU	● Medium	Aknowledged
OGGYINU.sol:344	Unexpected Ether Balance	OGGYINU	● Medium	Aknowledged
OGGYINU.sol:346	Unexpected Ether Balance	OGGYINU	● Medium	Aknowledged
OGGYINU.sol:375	Unauthenticated Storage Access	OGGYINU	● Medium	Aknowledged
OGGYINU.sol:376,385	Use Safer Functions	OGGYINU	● Low	Aknowledged
OGGYINU.sol:376	Prefer .call() To send()/transfer()	OGGYINU	● Informational	Aknowledged
OGGYINU.sol:3,5,34,161	Recommend to Follow Code Layout Conventions	ERC20	● Informational	Aknowledged
OGGYINU.sol:262,268,272,276,291,295,380,469	No Check of Address Params with Zero Address	OGGYINU	● Informational	Aknowledged

Location	Title	Scope	Severity	Status
OGGYINU.sol:218	Variables Should Be Constants	OGGYINU	● Informational	Aknowledged
OGGYINU.sol:161	No Need To Use SafeMath in Solidity Contract of Version 0.8.0 and Above	OGGYINU	● Informational	Aknowledged
OGGYINU.sol:146,404,421,423	Use Shift Operation Instead of Mul/Div	SafeMath	● Informational	Aknowledged
OGGYINU.sol:337	Unused Internal Function	OGGYINU	● Informational	Aknowledged
OGGYINU.sol:231,247,304,344,350,404	Cache State Variables that are Read Multiple Times within A Function	OGGYINU	● Informational	Aknowledged
OGGYINU.sol:354	Event Should be Emitted When Critical State Variables Change	OGGYINU	● Informational	Aknowledged
OGGYINU.sol:56,61,291,380,389,486	Function Visibility Can Be External	Ownable	● Informational	Aknowledged
OGGYINU.sol:52,62,128,136,147,155,285,305,312,334,381,394,395,396,464	Use CustomError Instead of String	Ownable	● Informational	Aknowledged
OGGYINU.sol:285	Lack of Error Message	OGGYINU	● Informational	Aknowledged
OGGYINU.sol:400	ReentrancyGuard Should Modify External Function	OGGYINU	● Informational	Aknowledged
OGGYINU.sol:62,147,312,396	Long String in revert/require	Ownable	● Informational	Aknowledged
OGGYINU.sol:164,215,216,217	Variables Can Be Declared as Immutable	OGGYINU	● Informational	Aknowledged

Location	Title	Scope	Severity	Status
OGGYINU.sol:376,419	Get Contract Balance of ETH in Assembly	OGGYINU	● Informational	Aknowledged
OGGYINU.sol:62	Use Assembly to Check Zero Address	Ownable	● Informational	Aknowledged
OGGYINU.sol:184	Too Many Digits	OGGYINU	● Informational	Aknowledged

Code Security - Unchecked Call Return Value

Title	Severity	Location	Status
Unchecked Call Return Value	● Medium	OGGYINU.sol:385,428, 430	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 - Unexpected Ether Balance

Title	Severity	Location	Status
Unexpected Ether Balance	● Medium	OGGYINU.sol:344	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 - Unexpected Ether Balance

Title	Severity	Location	Status
Unexpected Ether Balance	● Medium	OGGYINU.sol:346	Aknowledged

Description

Strict checks on the balance of the account may cause the contract to run abnormally, because the change of the balance of the account may be affected by various factors. For example, the selfdestruct method in a contract allows sending arbitrary Ether to another contract without triggering the fallback function of another contract.

Code Security - Unauthenticated Storage Access

Title	Severity	Location	Status
Unauthenticated Storage Access	🟠 Medium	OGGYINU.sol:375	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	OGGYINU.sol:376,385	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	OGGYINU.sol:376	Aknowledged

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	OGGYINU.sol:3,5,34,161	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 - No Check of Address Params with Zero Address

Title	Severity	Location	Status
No Check of Address Params with Zero Address	● Informational	OGGYINU.sol:262,268,272,276,291,295,380,469	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	$BART.sol:487,504,509	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 - Variables Should Be Constants

Title	Severity	Location	Status
Variables Should Be Constants	● Informational	OGGYINU.sol:218	Aknowledged

Description

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

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	OGGYINU.sol:161	Aknowledged

Description

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

Optimization Suggestion - Use Shift Operation Instead of Mul/Div

Title	Severity	Location	Status
Use Shift Operation Instead of Mul/Div	● Informational	OGGYINU.sol:146,404, 421,423	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 - Unused Internal Function

Title	Severity	Location	Status
Unused Internal Function	● Informational	OGGYINU.sol:337	Aknowledged

Description

Internal functions is defined but not used, which will add gas consumption.

Optimization Suggestion - Cache State Variables that are Read Multiple Times within A Function

Title	Severity	Location	Status
Cache State Variables that are Read Multiple Times within A Function	● Informational	OGGYINU.sol:231,247,304,344,350,404	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 - Event Should be Emitted When Critical State Variables Change

Title	Severity	Location	Status
Event Should be Emitted When Critical State Variables Change	● Informational	OGGYINU.sol:354	Aknowledged

Description

When some critical variables in the contract, such as owner and balance change, an event should be emitted so that the changes of these variables can be tracked off-chain.

Optimization Suggestion - Function Visibility Can Be External

Title	Severity	Location	Status
Function Visibility Can Be External	● Informational	OGGYINU.sol:56,61,291,380,389,486	Aknowledged

Description

Functions that are not called should be declared as external.

Optimization Suggestion - Use CustomError Instead of String

Title	Severity	Location	Status
Use CustomError Instead of String	● Informational	OGGYINU.sol:52,62,128,136,147,155,285,305,312,334,381,394,395,396,464	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	OGGYINU.sol:285	Aknowledged

Description

Use empty string as parameter while invoking function Revert() or Require().

Optimization Suggestion - ReentrancyGuard Should Modify External Function

Title	Severity	Location	Status
ReentrancyGuard Should Modify External Function	● Informational	OGGYINU.sol:400	Aknowledged

Description

The reentrancy guard modifier should modify the external function, because reentrancy vulnerabilities often occur in external calls.

Optimization Suggestion - Long String in revert/require

Title	Severity	Location	Status
Long String in revert/require	● Informational	OGGYINU.sol:62,147,312,396	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	OGGYINU.sol:164,215,216,217	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	OGGYINU.sol:376,419	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	OGGYINU.sol:62	Aknowledged

Description

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

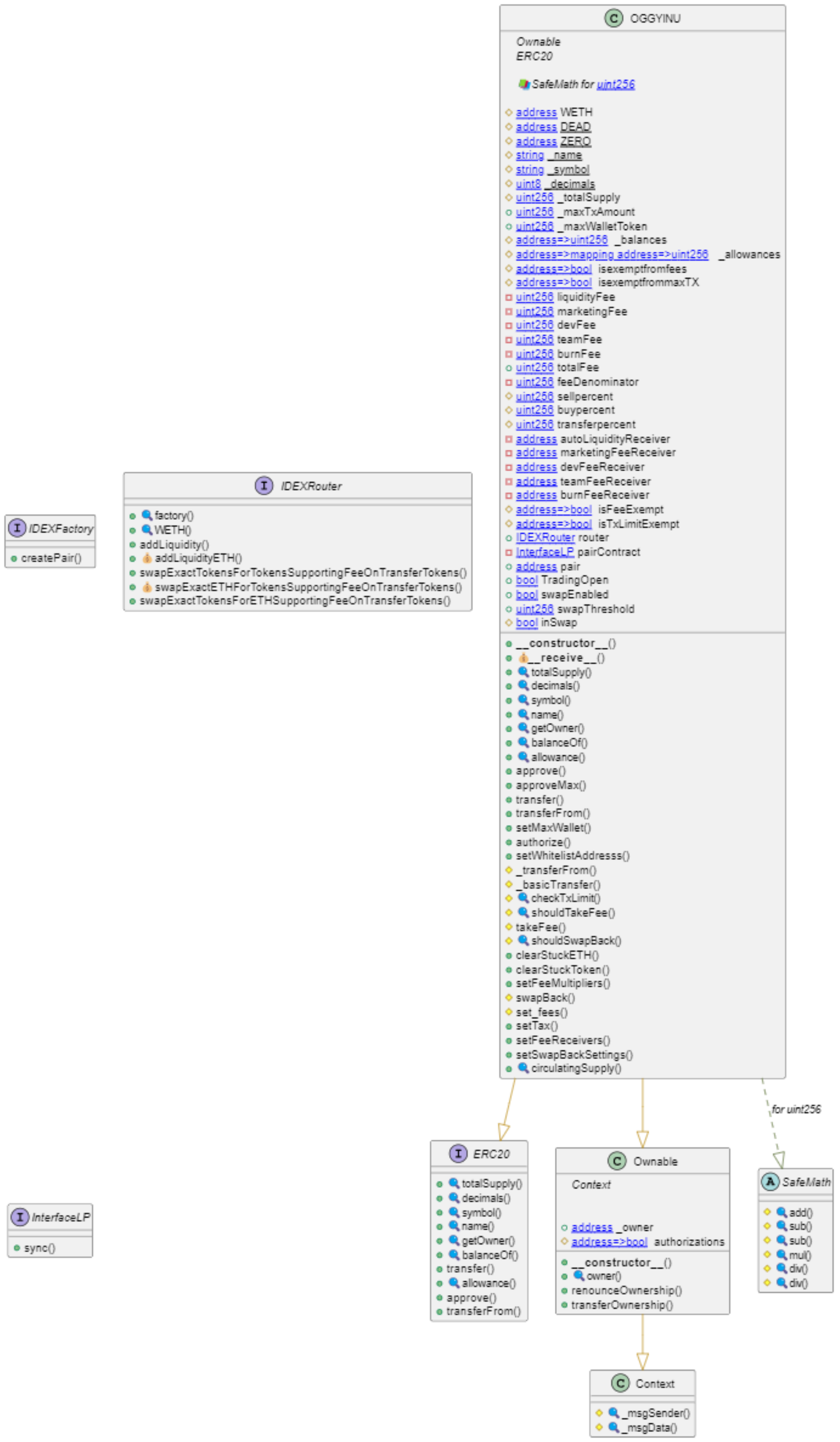
Optimization Suggestion - Too Many Digits

Title	Severity	Location	Status
Too Many Digits	● Informational	OGGYINU.sol:184	Aknowledged

Description

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

PlantUML


OGGYINU
Ownable
ERC20
SafeMath for uint256
address WETH
address DEAD
address ZERO
string _name
string _symbol
uint8 _decimals
uint256 _totalSupply
uint256 _maxTxAmount
uint256 _maxWalletToken
address=>uint256 _balances
address=>mapping address=>uint256 _allowances
address=>bool isexemptfromfees
address=>bool isexemptfrommaxTX
uint256 liquidityFee
uint256 marketingFee
uint256 devFee
uint256 teamFee
uint256 burnFee
uint256 totalFee
uint256 feeDenominator
uint256 sellpercent
uint256 buypercent
uint256 transferpercent
address autoLiquidityReceiver
address marketingFeeReceiver
address devFeeReceiver
address teamFeeReceiver
address burnFeeReceiver
address=>bool isFeeExempt
address=>bool isTxLimitExempt
IDEXRouter router
InterfaceLP pairContract
address pair
bool TradingOpen
bool swapEnabled
uint256 swapThreshold
bool inSwap
__constructor__()
__receive__()
totalSupply()
decimals()
symbol()
name()
getOwner()
balanceOf()
allowance()
approve()
approveMax()
transfer()
transferFrom()
setMaxWallet()
authorize()
setWhitelistAddresss()
_transferFrom()
_basicTransfer()
checkTxLimit()
shouldTakeFee()
takeFee()
shouldSwapBack()
clearStuckETH()
clearStuckToken()
setFeeMultipliers()
swapBack()
set_fees()
setTax()
setFeeReceivers()
setSwapBackSettings()
circulatingSupply()
IDEXFactory
createPair()
IDEXRouter
factory()
WETH()
addLiquidity()
addLiquidityETH()
swapExactTokensForTokensSupportingFeeOnTransferTokens()
swapExactETHForTokensSupportingFeeOnTransferTokens()
swapExactTokensForETHSupportingFeeOnTransferTokens()
InterfaceLP
sync()
ERC20
totalSupply()
decimals()
symbol()
name()
getOwner()
balanceOf()
transfer()
allowance()
approve()
transferFrom()
Ownable
Context
address _owner
address=>bool authorizations
__constructor__()
owner()
renounceOwnership()
transferOwnership()
for uint256
SafeMath
add()
sub()
sub()
mul()
div()
div()
Context
_msgSender()
_msgData()

Appendix

Finding Categories

Security and Best Practices

1. Unchecked Call Return Value: Smart contracts should rigorously check the return values of external calls to prevent vulnerabilities.
2. Unexpected Ether Balance: Regularly monitor the contract's ether balance to detect any unexpected or unauthorized changes.
3. Unauthenticated Storage Access: Conduct thorough reviews to identify and rectify instances of unauthenticated storage access, which can lead to unauthorized data tampering.
4. Use Safer Functions: Secure your contracts by using well-established and secure functions known for their robust design.
5. Prefer .call() To send()/transfer(): Implement the use of .call() for external contract calls to reduce security risks, avoiding send()/transfer().
6. Recommend to Follow Code Layout Conventions: Consistent adherence to code layout conventions can greatly enhance code readability and maintainability.
7. No Check of Address Params with Zero Address: Verify address parameters to include checks ensuring that the address is not the zero address.
8. Variables Should Be Constants: Consider marking variables as constants if they should not change after initialization.
9. No Need To Use SafeMath in Solidity Contract of Version 0.8.0 and Above: Solidity versions 0.8.0 and above include built-in overflow and underflow protection, reducing the reliance on the SafeMath library.
10. Use Shift Operation Instead of Mul/Div: Enhance efficiency by employing shift operations instead of traditional multiplication and division.
11. Unused Internal Function: Remove or refactor unused internal functions to reduce contract complexity.
12. Cache State Variables that are Read Multiple Times within A Function: Improve efficiency by caching state variables that are accessed multiple times within a function.
13. Event Should be Emitted When Critical State Variables Change: Emit events to notify external applications when significant state variable changes occur.
14. Function Visibility Can Be External: Optimize gas consumption by setting functions to external visibility when they are accessed only from within the contract.
15. Use CustomError Instead of String: Consider using custom error codes instead of string error messages to optimize contract performance.
16. Lack of Error Message: Always provide descriptive error messages to improve contract debugging and user experience.
17. ReentrancyGuard Should Modify External Function: Ensure that ReentrancyGuard is used to modify external functions to prevent reentrancy attacks.
18. Long String in revert/require: Minimize gas usage by avoiding long revert or require error messages and optimizing for gas efficiency.
19. Variables Can Be Declared as Immutable: Declare variables as immutable if they do not change after initialization, enhancing security and readability.
20. Get Contract Balance of ETH in Assembly: Use optimized assembly to efficiently check the contract's ETH balance.
21. Use Assembly to Check Zero Address: Employ optimized assembly checks to efficiently verify zero addresses.
22. Too Many Digits: Avoid excessive decimal places or precision that may lead to unexpected issues.
23. Secure Project Management: Adhere to best practices in project management to ensure secure and efficient development processes.
24. Code Documentation: Comprehensive code documentation is essential for team collaboration and future code maintenance.
25. Trusted Sources for External Contracts: Ensure that external contracts are sourced from reputable and verified developers.
26. Regular Code Audits: Perform routine code audits to identify and address 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://oggyinu.com/

Network Security

High | 0 Attentions

Application Security

High | 6 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

✓ 7 Passed

ⓘ 6 Attention

Missing X-Frame-Options Header	YES ⓘ
Missing HSTS header	YES ⓘ
Missing X-Content-Type-Options Header	YES ⓘ
Missing Content Security Policy (CSP)	YES ⓘ
HTTP Access Allowed	NO ✓
Self-Signed Certificate	NO ✓
Wrong Host Certificate	NO ✓
Expired Certificate	NO ✓
SSL/TLS Supports Weak Cipher	YES ⓘ
Support SSL Protocols	NO ✓
Support TLS Weak Version	YES ⓘ

DNS Health

✓ 7 Passed

i 3 Attention

Missing SPF Record	YES	i
Missing DMARC Record	YES	i
Missing DKIM Record	YES	i
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

✓ 4 Passed

ⓘ 6 Failed

X (Twitter)	↗	PASS ✓
Facebook		FAIL ✗
Instagram		FAIL ✗
TikTok	↗	PASS ✓
YouTube	↗	PASS ✓
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

Somewhat Developed	MEDIUM
Token Launch Date	2023.10.23 15:15 (UTC)
Token Market Cap (estimate)	$3.01M
Token/Project Age	34 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	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.