SPHINXSHIELD

Security Assessment

MAGA

Mar 3th, 2024

Evaluation Outcomes

Security Score

Review	Score
Overall Score	83/100
Auditor Score	81/100

Review by Section	Score
Manual Scan Score	48/57
Advance Check Score	14/19

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

[illegible]

Table of Contents

Summary

This audit report is tailored for **MAGA** aiming to uncover potential issues and vulnerabilities within the **MAGA** 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	MAGA
Blockchain	Ethereum
Language	Solidity
Codebase	https://etherscan.io/token/0x576e2BeD8F7b46D34016198911Cdf9886f78bea7
Commit	5671b2f00201592e4aa66fe90f1d595ff624c90824bd4d80c4aee39730be3727

Audit Summary

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

Vulnerability Summary

24 Total Findings

0 Resolved

0 Mitigated

0 Partially Resolved

24 Aknowledged

0 Declined

Vulnerability Level	Total	Pending	Declined	Aknowledged	Resolved
● High	2	0	0	2	0
● Medium	6	0	0	6	0
● Low	1	0	0	1	0
● Informational	15	0	0	15	0
● Discussion	0	0	0	0	0

Audit Scope

ID	File	KECCAK256 or SHA256 Checksum
MAG	MAGA.sol	0x2fcf4191c2e4e9f62b610af5147972bc3a17b3d4a0ddf1fee09c0d7f162a489f

Understandings

MAGA is an ERC20 token deployed on the Ethereum blockchain. It implements a taxation mechanism on buys, sells, and transfers to support U.S. veterans and child rescue initiatives. Here's a brief overview:

Token Information

- Name: MAGA
- Symbol: TRUMP
- Decimals: 9
- Total Supply: 47,000,000 TRUMP

Taxation Mechanism

- (1%) Tax on buys, sells, and transfers.
- (0.33%) Allocation to U.S. veterans and child rescue donations.
- (0.66%) Allocated for marketing, development, and liquidity.

Ownership and Authorization

The contract owner can adjust tax parameters and authorize specific addresses for privileged functions.

Transaction Limits

Transaction limits are enforced to manage token movement effectively and prevent excessive transactions.

Swap Mechanism

Utilizes a swap mechanism to manage liquidity, with thresholds and conditions for efficient swapping.

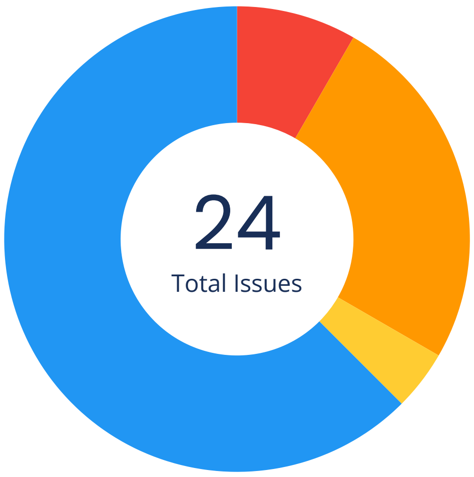
Additional Functionality

Includes functions for clearing stuck tokens and manual swaps to ensure smooth contract operation.

About MAGA

Please note that the MAGA contract is a vital component of the MAGA Movement, facilitating its goals of supporting veterans and protecting children.

Findings



- ● High - 2
- ● Medium - 6
- ● Low - 1
- ● Informational - 15
- ● Discussion - 0

Location	Title	Scope	Severity	Status
MAGA.sol:305	Arbitrary External Call	MAGA	● High	Aknowledged
MAGA.sol:325	Reentrancy	MAGA	● High	Aknowledged
MAGA.sol:331	Unchecked Call Return Value	MAGA	● Medium	Aknowledged
MAGA.sol:187	Unauthenticated Storage Access	MAGA	● Medium	Aknowledged
MAGA.sol:196	Unauthenticated Storage Access	MAGA	● Medium	Aknowledged
MAGA.sol:201	Unauthenticated Storage Access	MAGA	● Medium	Aknowledged
MAGA.sol:308	Unauthenticated Storage Access	MAGA	● Medium	Aknowledged
MAGA.sol:314	Unauthenticated Storage Access	MAGA	● Medium	Aknowledged

Location	Title	Scope	Severity	Status
MAGA.sol:296,297, 305,311,331	Use Safer Functions	MAGA	🟡 Low	Aknowledged
MAGA.sol:296,297, 311	Prefer .call() To send()/transfer()	MAGA	🔵 Informational	Aknowledged
MAGA.sol:3,11,60,1 10	Recommend to Follow Code Layout Conventions	IERC20	🔵 Informational	Aknowledged
MAGA.sol:147	Unused Events	MAGA	🔵 Informational	Aknowledged
MAGA.sol:300	No Check of Address Params with Zero Address	MAGA	🔵 Informational	Aknowledged
MAGA.sol:117,118, 122,123,124,125,12 6,127,128,137,138	Variables Should Be Constants	MAGA	🔵 Informational	Aknowledged
MAGA.sol:110	No Need To Use SafeMath in Solidity Contract of Version 0.8.0 and Above	MAGA	🔵 Informational	Aknowledged
MAGA.sol:217,244, 269,320	Use != 0 Instead of > 0 for Unsigned Integer Comparison	MAGA	🔵 Informational	Aknowledged
MAGA.sol:79	Function Visibility Can Be External	Ownable	🔵 Informational	Aknowledged
MAGA.sol:25,34,75, 208,209,215,216,21 7,223,224,227,233, 309,326	Use CustomError Instead of String	Ownable	🔵 Informational	Aknowledged
MAGA.sol:227	Lack of Error Message	MAGA	🔵 Informational	Aknowledged
MAGA.sol:272	ReentrancyGuard Should Modify External Function	MAGA	🔵 Informational	Aknowledged

Location	Title	Scope	Severity	Status
MAGA.sol:159,220, 222,224,287,328	Cache State Variables That Are Read Multiple Times within a Function	MAGA	● Informational	Aknowledged
MAGA.sol:115,116	Variables Can Be Declared as Immutable	MAGA	● Informational	Aknowledged
MAGA.sol:243,245, 309,310,319	Get Contract Balance of ETH in Assembly	MAGA	● Informational	Aknowledged
MAGA.sol:208,209, 215,216	Use Assembly to Check Zero Address	MAGA	● Informational	Aknowledged

Code Security - Arbitrary External Call

Title	Severity	Location	Status
Arbitrary External Call	🔴 High	MAGA.sol:305	Aknowledged

Description

If the address of an external call or transfer can be controlled by the user, the user can perform arbitrary actions to cause the attack to occur.

Code Security - Reentrancy

Title	Severity	Location	Status
Reentrancy	🔴 High	MAGA.sol:325	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	MAGA.sol:331	Aknowledged

Description

The return value of low level calls and external calls (transfer, transferFrom and approve) should be verified since low level calls may fail and these three external function calls may only return false but not cause execution reverted once fail. If not properly handled, it might incur asset losses to users and the project party.

Code Security - Unauthenticated Storage Access

Title	Severity	Location	Status
Unauthenticated Storage Access	● Medium	MAGA.sol:187	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	MAGA.sol:196	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	MAGA.sol:201	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	MAGA.sol:308	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	MAGA.sol:314	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	MAGA.sol:296,297,305 ,311,331	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	MAGA.sol:296,297,311	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	MAGA.sol:3,11,60,110	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	MAGA.sol:147	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	MAGA.sol:300	Aknowledged

Description

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

Optimization Suggestion - Variables Should Be Constants

Title	Severity	Location	Status
Variables Should Be Constants	● Informational	MAGA.sol:117,118,122 ,123,124,125,126,127, 128,137,138	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	MAGA.sol:110	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 != 0 Instead of > 0 for Unsigned Integer Comparison

Title	Severity	Location	Status
Use != 0 Instead of > 0 for Unsigned Integer Comparison	● Informational	MAGA.sol:217,244,269,320	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	MAGA.sol:79	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	MAGA.sol:25,34,75,208,209,215,216,217,223,224,227,233,309,326	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	MAGA.sol:227	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	MAGA.sol:272	Aknowledged

Description

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

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	MAGA.sol:159,220,222 ,224,287,328	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	MAGA.sol:115,116	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	MAGA.sol:243,245,309,310,319	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.

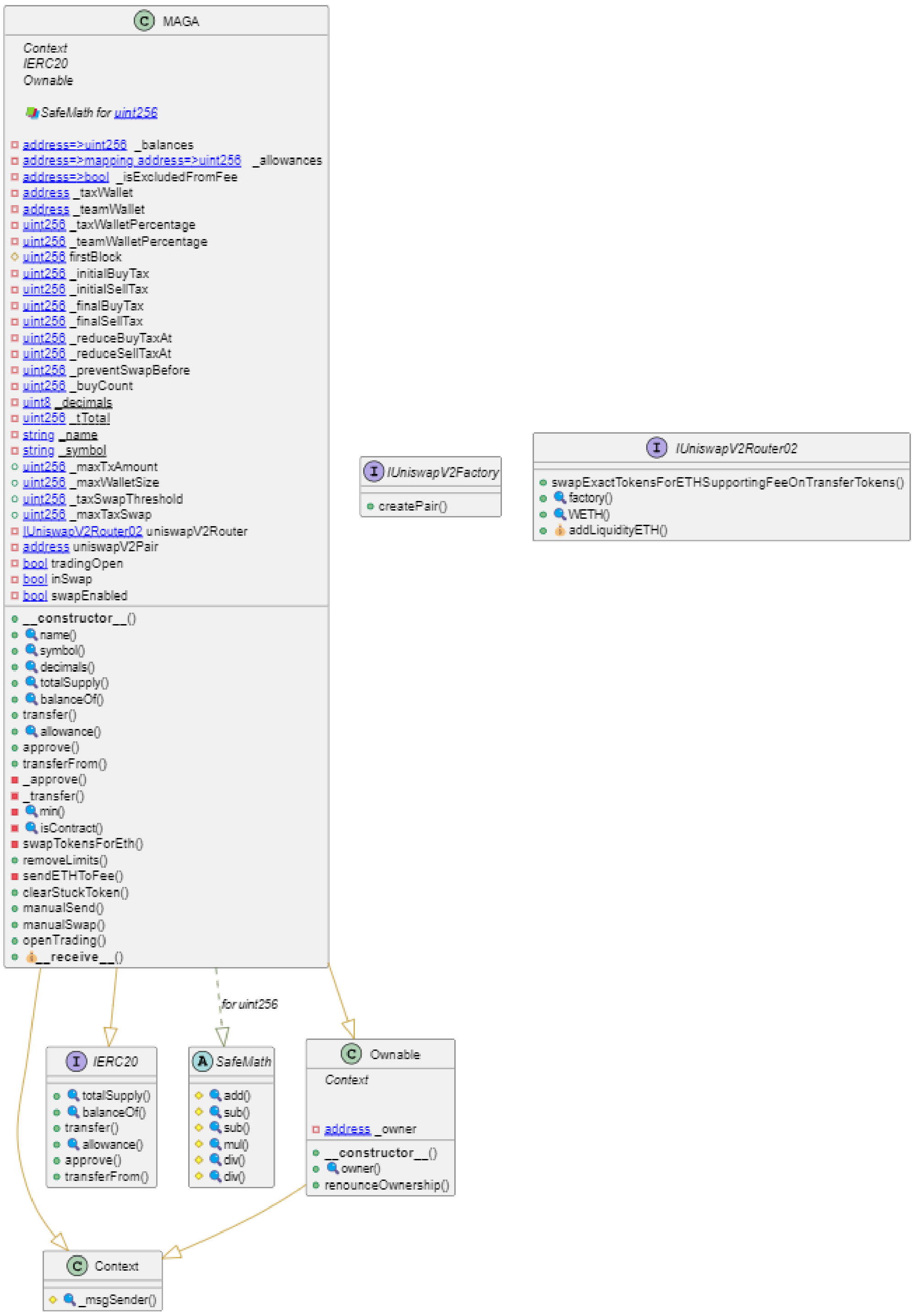
Optimization Suggestion - Use Assembly to Check Zero Address

Title	Severity	Location	Status
Use Assembly to Check Zero Address	● Informational	MAGA.sol:208,209,215,216	Aknowledged

Description

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

PlantUML


MAGA
Context
IERC20
Ownable
SafeMath for uint256
address=>uint256 _balances
address=>mapping address=>uint256 _allowances
address=>bool _isExcludedFromFee
address _taxWallet
address _teamWallet
uint256 _taxWalletPercentage
uint256 _teamWalletPercentage
uint256 firstBlock
uint256 _initialBuyTax
uint256 _initialSellTax
uint256 _finalBuyTax
uint256 _finalSellTax
uint256 _reduceBuyTaxAt
uint256 _reduceSellTaxAt
uint256 _preventSwapBefore
uint256 _buyCount
uint8 _decimals
uint256 _tTotal
string _name
string _symbol
uint256 _maxTxAmount
uint256 _maxWalletSize
uint256 _taxSwapThreshold
uint256 _maxTaxSwap
IUniswapV2Router02 uniswapV2Router
address uniswapV2Pair
bool tradingOpen
bool inSwap
bool swapEnabled
__constructor__()
name()
symbol()
decimals()
totalSupply()
balanceOf()
transfer()
allowance()
approve()
transferFrom()
_approve()
_transfer()
min()
isContract()
swapTokensForEth()
removeLimits()
sendETHToFee()
clearStuckToken()
manualSend()
manualSwap()
openTrading()
__receive__()
IUniswapV2Factory
createPair()
IUniswapV2Router02
swapExactTokensForETHSupportingFeeOnTransferTokens()
factory()
WETH()
addLiquidityETH()
for uint256
IERC20
totalSupply()
balanceOf()
transfer()
allowance()
approve()
transferFrom()
SafeMath
add()
sub()
sub()
mul()
div()
div()
Ownable
Context
address _owner
__constructor__()
owner()
renounceOwnership()
Context
_msgSender()

Appendix

Finding Categories

Security and Best Practices

1. Arbitrary External Call: Avoid arbitrary external calls within smart contracts to mitigate potential security risks associated with untrusted code execution.
2. Reentrancy: Implement proper reentrancy protection mechanisms to prevent recursive calls and ensure contract state integrity.
3. Unchecked Call Return Value: Validate return values from external calls to prevent exploitation of unchecked return values.
4. Unauthenticated Storage Access: Scrutinize smart contracts for unauthenticated storage access vulnerabilities, which can lead to unauthorized data manipulation.
5. Use Safer Functions: Opt for functions with known secure design patterns to minimize the risk of security vulnerabilities.
6. Prefer .call() To send()/transfer(): Use .call() over send()/transfer() for external contract calls to reduce security risks associated with unexpected contract behavior.
7. Recommend to Follow Code Layout Conventions: Adhere to established code layout conventions to enhance code readability and maintainability.
8. Unused Events: Remove unused events from smart contracts to reduce contract size and complexity, improving overall contract efficiency.
9. No Check of Address Params with Zero Address: Ensure address parameters are checked to prevent potential vulnerabilities related to zero addresses.
10. Variables Should Be Constants: Declare variables as constants where applicable to enhance code clarity and prevent unintentional variable modifications.
11. No Need To Use SafeMath in Solidity Contract of Version 0.8.0 and Above: Leverage built-in overflow and underflow protection in Solidity versions 0.8.0 and above, eliminating the need for SafeMath library usage.
12. Use != 0 Instead of > 0 for Unsigned Integer Comparison: Utilize != 0 instead of > 0 for unsigned integer comparison to mitigate potential vulnerabilities related to integer overflow.
13. Function Visibility Can Be External: Enhance gas efficiency by setting function visibility to external if they are only accessed within the contract.
14. Use CustomError Instead of String: Employ custom error codes instead of string error messages for improved contract efficiency and gas optimization.
15. Lack of Error Message: Provide informative error messages to users to enhance contract usability and facilitate debugging.
16. ReentrancyGuard Should Modify External Function: Ensure that ReentrancyGuard modifiers are applied to external functions to prevent reentrancy attacks.
17. 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 optimize contract performance.
18. Variables Can Be Declared as Immutable: Declare variables as immutable if they do not change after initialization to improve contract security and readability.
19. Get Contract Balance of ETH in Assembly: Optimize gas usage by retrieving contract ETH balance using assembly to reduce overhead costs.
20. Use Assembly to Check Zero Address: Employ optimized assembly checks to verify zero addresses efficiently and securely.

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://magamemecoin.com/

Network Security

High | 0 Attentions

Application Security

High | 3 Attentions

DNS Security

High | 4 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 | ⓘ 3 Attention

Missing X-Frame-Options Header	YES ⓘ
Missing HSTS header	NO ✓
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	NO ✓
Support SSL Protocols	NO ✓
Support TLS Weak Version	NO ✓

DNS Health

✓ 7 Passed

ℹ 4 Attention

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

Social Media Checks

✓ 2 Passed   i 8 Failed

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

Recommendation

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

Social Media Information Notes

Unspecified Auditor Notes

Notes from the Project Owner

Fundamental Health

KYC Status

SphinxShield KYC	NO ⚠
3rd Party KYC	NO ✖

Project Maturity Metrics

Developing	**Low-Medium**
Token Launch Date	**2023.08.11 23:15 (UTC)**
Token Market Cap (estimate)	**$379.98M**
Token/Project Age	**204 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 BitMart and CoinEx.

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.