

# SPHINXSHIELD

## Security Assessment

## **Superman Coin**

## Oct 15th, 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	87/100
Auditor Score	83/100

Review by Section	Score
Manual Scan Score	23/57
Advance Check Score	13/19

## Scoring System

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

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

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





# Table of Contents

## **Summary**

### **Overview**

[Project Summary](#)

[Audit Summary](#)

[Vulnerability Summary](#)

[Audit Scope](#)

### **Findings**

### **PlantUML**

### **Appendix**

### **Website Scan**

### **Social Media Checks**

### **Fundamental Health**

### **Disclaimer**

### **About**



# Summary

This audit report is tailored for **Superman Coin**, aiming to uncover potential issues and vulnerabilities within the **Superman Coin** 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	Supermain Coin
Blockchain	Ethereum
Language	Solidity
Codebase	<a href="https://etherscan.io/address/0x3A548fc09ad72bCf2a2f8a5753C182d242bA89aE">https://etherscan.io/address/0x3A548fc09ad72bCf2a2f8a5753C182d242bA89aE</a>
Commit	

## Audit Summary

Delivery Date	Oct 15th, 2023
Audit Methodology	Static Analysis, Manual Review
Key Components	

## Vulnerability Summary

Vulnerability Level	Total	⚠ Pending	⊗ Declined	ℹ Acknowledged	✓ Resolved
● High	1	0	0	1	0
● Medium	1	0	0	1	0
● Low	2	0	0	2	0
● Informational	20	0	0	20	0
● Discussion	0	0	0	0	0

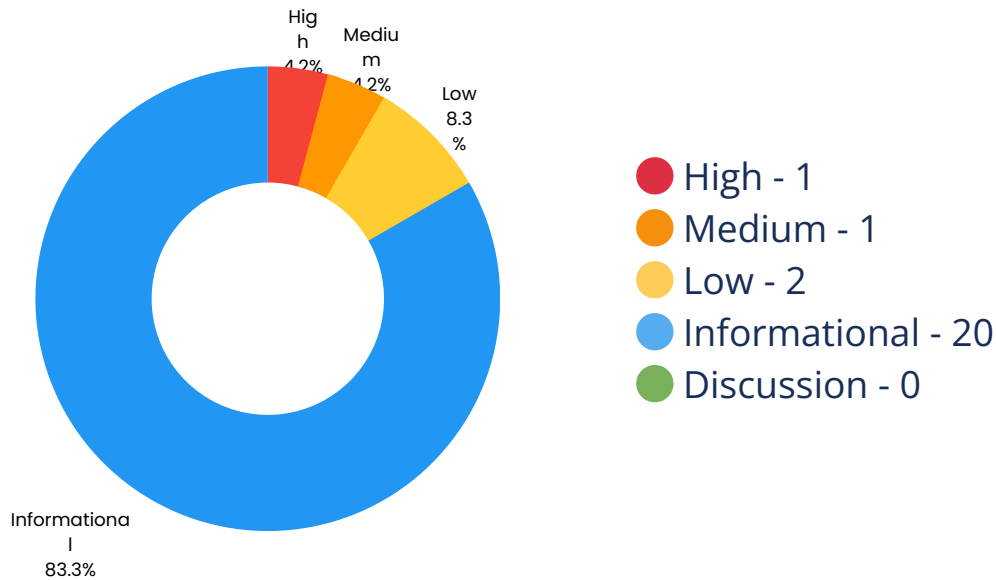


# Audit Scope

ID	File	KECCAK256 or SHA256 Checksum
SPM	Superman.sol	0x1dce26b7e2deaaa7175e418cebd043427d090c43ff72f45ba9fa4ea8a46541fc



# Findings



Location	Title	Scope	Severity	Status
Superman.sol:1091	Reentrancy	Superman	High	Acknowledged
Superman.sol:1121	Unchecked Call Return Value	Superman	Medium	Acknowledged
Superman.sol:902	Uninitialized Variables	Superman	Low	Acknowledged
Superman.sol:986	Use Safer Functions	Superman	Low	Acknowledged
Superman.sol:760, 6,40,895	Recommend to Follow Code Layout Conventions	Ownable	Informational	Acknowledged
Superman.sol:1133, 1132	Parameters Should Be Declared as Calldata	Superman	Informational	Acknowledged
Superman.sol:995, 971,964,978	No Check of Address Params with Zero Address	Superman	Informational	Acknowledged
Superman.sol:901	Variables Should Be Constants	Superman	Informational	Acknowledged
Superman.sol:895	No Need To Use SafeMath in Solidity Contract of Version 0.8.0 and Above	Superman	Informational	Acknowledged
Superman.sol:1137	Use ++i/--i Instead of i++/i--	Superman	Informational	Acknowledged



Location	Title	Scope	Severity	Status
Superman.sol:1112,1093	Cache State Variables that are Read Multiple Times within A Function	Superman	● Informational	Acknowledged
Superman.sol:1007,995,249,298,320,343,990,224,363,256,275,978,1091,232	Function Visibility Can Be External	ERC20	● Informational	Acknowledged
Superman.sol:366,90,394,477,420,958,393,499,399,1136,71,478,1135	Use CustomError Instead of String	Ownable	● Informational	Acknowledged
Superman.sol:901	Unused State Variables	Superman	● Informational	Acknowledged
Superman.sol:393,366,90,394,399,1136,477,478	Long String in revert/require	Ownable	● Informational	Acknowledged
Superman.sol:985	Get Contract Balance of ETH in Assembly	Superman	● Informational	Acknowledged
Superman.sol:540,520	Empty Function Body	ERC20	● Informational	Acknowledged
Superman.sol:393,90,394,477,420,478	Use Assembly to Check Zero Address	Ownable	● Informational	Acknowledged
Superman.sol:986	Prefer .call() To send()/transfer()	Superman	● Informational	Acknowledged
Superman.sol:958	Lack of Error Message	Superman	● Informational	Acknowledged
Superman.sol:898	Variables Can Be Declared as Immutable	Superman	● Informational	Acknowledged
Superman.sol:900	Variables Can Be Declared as Immutable	Superman	● Informational	Acknowledged
Superman.sol:911	Variables Can Be Declared as Immutable	Superman	● Informational	Acknowledged





Location	Title	Scope	Severity	Status
Superman.sol:912	Variables Can Be Declared as Immutable	Superman	<div><div></div>Informational</div>	Aknowledged



## Code Security – Reentrancy

Title	Severity	Location	Status
Reentrancy	● High	Superman.sol:1091	Aknowledged

### Description

Step 1:

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.

Steps: 2:

Storage modification

### Recommendation

Optimization Suggestion - Function Visibility Can Be External

uint256 tokens

Parameter: tokens

Function: forceSwapAndSendDividends

Contract: Superman

Type Info:

uint256

(uint256) Argument forceSwapAndSendDividends.tokens



## Code Security – Unchecked Call Return Value

Title	Severity	Location	Status
Unchecked Call Return Value	● Medium	Superman.sol:1121	Acknowledged

### Description

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

## Code Security – Uninitialized Variables

Title	Severity	Location	Status
Uninitialized Variables	● Low	Superman.sol:902	Acknowledged

### Description

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

## Code Security – Use Safer Functions

Title	Severity	Location	Status
Use Safer Functions	● Low	Superman.sol:986	Acknowledged

### 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 - Recommend to Follow Code Layout Conventions

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

Recommend to Follow Code Layout Conventions

● Informational

Superman.sol:760,6,4  
0,895

Acknowledged

### Description

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

## Optimization Suggestion - Parameters Should Be Declared as Calldata

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

Parameters Should Be Declared as Calldata

● Informational

Superman.sol:1133,11  
32

Acknowledged

### Description

When the compiler parses the external or public function, it can directly read the function parameters from calldata. Setting it to other storage locations may waste gas. About 300-400 gas can be saved with optimization turned off while 120-150 gas can be saved vice versa.

## Optimization Suggestion - No Check of Address Params with Zero Address

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

No Check of Address Params with Zero Address

● Informational

Superman.sol:995,971  
,964,978

Acknowledged

### Description

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



## Optimization Suggestion – Unused State Variables

Title	Severity	Location	Status
Variables Should Be Constants	<span>●</span> Informational	Superman.sol:901	Aknowledged

### Description

The state variables are not used, which will increase the gas consumption.

## 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	<span>●</span> Informational	Superman.sol:895	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 ++i/--i Instead of i++/i--

Title	Severity	Location	Status
Use ++i/--i Instead of i++/i--	<span>●</span> Informational	Superman.sol:1137	Aknowledged

### Description

Compared with i++, ++i can save about 5 gas per use. Compared with i--, --i can save about 3 gas per use in for loop.



## 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

Superman.sol:1112,1093

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 – Function Visibility Can Be External

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

Function Visibility Can Be External

● Informational

Superman.sol:1007,995,249,298,320,343,990,224,363,256,275,978,1091,232

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

Superman.sol:366,90,394,477,420,958,393,499,399,1136,71,478,1135

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 – Unused State Variables

Title	Severity	Location	Status
Unused State Variables	<span>●</span> Informational	Superman.sol:901	Aknowledged

### Description

The state variables are not used, which will increase the gas consumption.

## Optimization Suggestion – Use CustomError Instead of String

Title	Severity	Location	Status
Long String in revert/require	<span>●</span> Informational	Superman.sol:393,366 ,90,394,399,1136,477, 478	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 – Get Contract Balance of ETH in Assembly

Title	Severity	Location	Status
Get Contract Balance of ETH in Assembly	<span>●</span> Informational	Superman.sol:985	Aknowledged

### Description

Using the selfbalance and balance opcodes to get the ETH balance of the contract in assembly saves gas compared to getting the ETH balance through address(this).balance and xx.balance. When compiler optimization is turned off, about 210-250 gas can be saved, and when compiler optimization is turned on, about 50-100 gas can be saved.



## Optimization Suggestion – Empty Function Body

Title	Severity	Location	Status
Empty Function Body	<span>●</span> Informational	Superman.sol:540,520	Aknowledged

### Description

The body of this function is empty.

## Optimization Suggestion – Long String in revert/require

Title	Severity	Location	Status
Use Assembly to Check Zero Address	<span>●</span> Informational	Superman.sol:393,90, 394,477,420,478	Aknowledged

### Description

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

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

Title	Severity	Location	Status
Prefer .call() To send()/transfer()	<span>●</span> Informational	Superman.sol:986	Aknowledged

### Description

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





## Optimization Suggestion - Variables Can Be Declared as Immutable

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

Variables Can Be Declared as Immutable	<span>●</span> Informational	Superman.sol:898	Acknowledged
--	------------------------------	------------------	--------------

### 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 - Variables Can Be Declared as Immutable

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

Variables Can Be Declared as Immutable	<span>●</span> Informational	Superman.sol:900	Acknowledged
--	------------------------------	------------------	--------------

### 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 - Variables Can Be Declared as Immutable

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

Variables Can Be Declared as Immutable	<span>●</span> Informational	Superman.sol:911	Acknowledged
--	------------------------------	------------------	--------------

### 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 - Variables Can Be Declared as Immutable

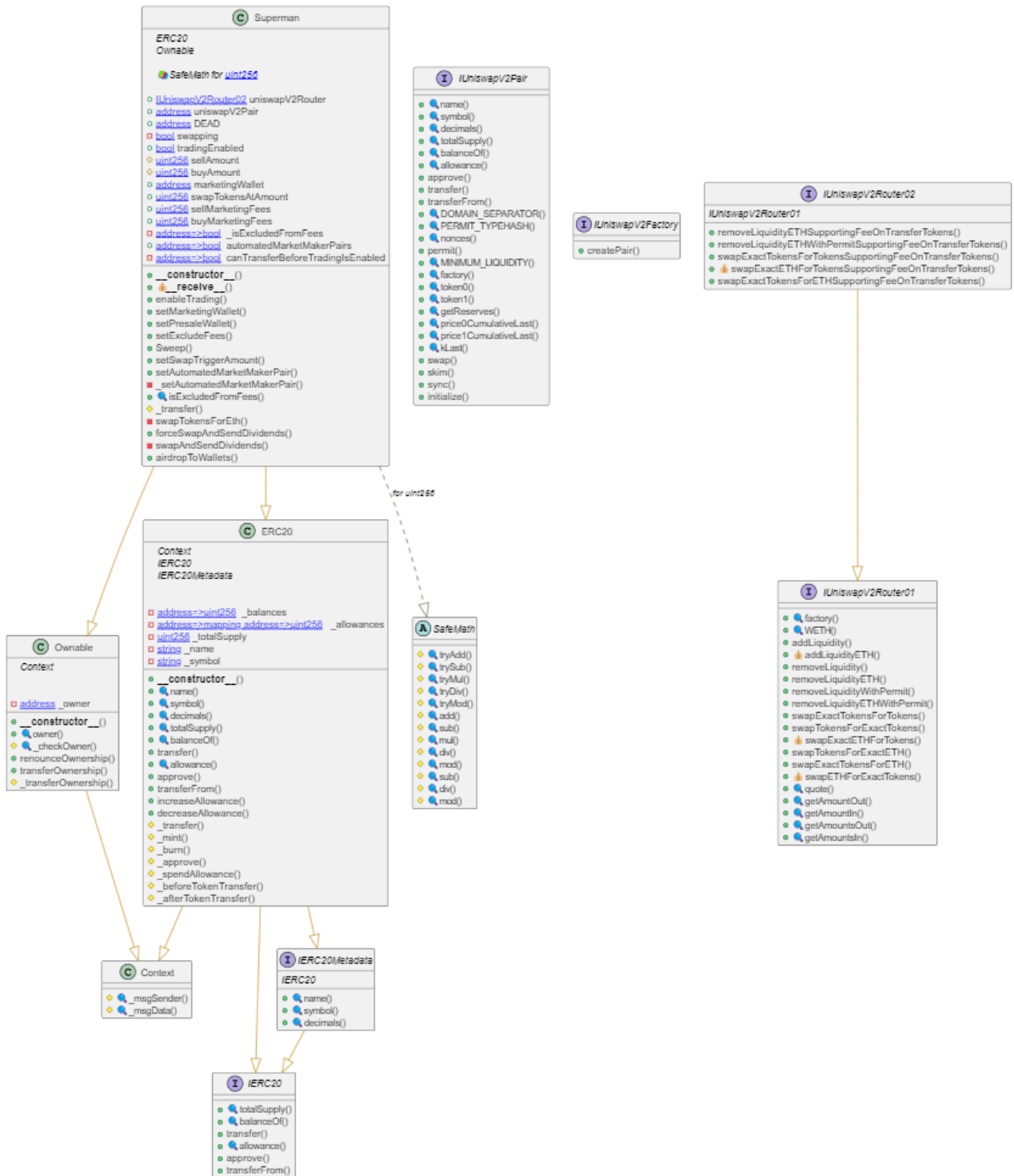
Title	Severity	Location	Status
Variables Can Be Declared as Immutable	<span>●</span> Informational	Superman.sol:912	Aknowledged

### 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.



# PlantUML





# Appendix

## Finding Categories

### Security and Best Practices

- 1.Reentrancy: The contract should be inspected for potential reentrancy vulnerabilities. A secure locking mechanism should be employed to mitigate this risk.
- 2.Unchecked Call Return Value: Smart contracts must validate the return value of external function calls. Unchecked return values can lead to unexpected behaviors.
- 3.Uninitialized Variables: Be diligent in initializing variables before use. Uninitialized variables can introduce inconsistencies into the contract state.
- 4.Use Safer Functions: Securely designed functions should be used to minimize security vulnerabilities. Review functions for potential security improvements.
- 5.Recommend to Follow Code Layout Conventions: Adhering to established code layout conventions enhances code readability and maintainability.
- 6.Parameters Should Be Declared as Calldata: Immutable parameters that do not need to be modified should be declared as calldata for added security.
- 7.No Check of Address Params with Zero Address: Ensure that checks for address parameters include verification that the address is not the zero address.
- 8.Variables Should Be Constants: Variables that do not change during the contract's execution should be declared as constants.
- 9.No Need To Use SafeMath in Solidity Contract of Version 0.8.0 and Above: Solidity versions 0.8.0 and above incorporate built-in overflow and underflow protection, reducing the need for SafeMath library usage.
- 10.Use ++i/--i Instead of i++/i--: Pre-increment and pre-decrement operators (++i/--i) should be preferred over post-increment and post-decrement operators (i++/i--) to avoid potential optimization issues.
- 11.Cache State Variables that are Read Multiple Times within A Function: For state variables read multiple times within a function, caching their values can improve gas efficiency.
- 12.Function Visibility Can Be External: Functions accessible only from the contract can be made external for improved gas efficiency.
- 13.Use CustomError Instead of String: Custom error codes are preferable to string error messages for enhanced contract efficiency.
- 14.Unused State Variables: Remove any unused state variables to maintain a cleaner codebase.
- 15.Long String in revert/require: Long revert or require strings can increase gas usage and should be optimized for gas efficiency.
- 16.Get Contract Balance of ETH in Assembly: Optimized assembly code can be employed to get the contract's ETH balance more efficiently.
- 17.Empty Function Body: Ensure functions do not have empty bodies, which can introduce vulnerabilities.
- 18.Use Assembly to Check Zero Address: Assemble checks can be utilized to verify zero addresses efficiently.
- 19.Prefer .call() To send()/transfer(): Use .call() for making external contract calls to minimize security risks.
- 20.Lack of Error Message: Include clear and informative error messages in require and revert statements for improved contract usability.
- 21.Variables Can Be Declared as Immutable: Variables that do not change after initialization can be declared as immutable for enhanced security and readability.



## 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://supermancoin.finance/>



Network Security

High | 0 Attentions

Application Security

High | 0 Attentions










DNS Security

High | 0 Attentions

Network Security

 **9 Passed**

 **0 Attention**

FTP Service Anonymous LOGIN	NO	
VNC Service Accesible	NO	
RDP Service Accesible	NO	
LDAP Service Accesible	NO	
PPTP Service Accesible	NO	
RSYNC Service Accesible	NO	
SSH Weak Cipher	NO	
SSH Support Weak MAC	NO	
CVE on the Related Service	NO	



## Application Security

✓ 7 Passed

i 4 Attention

Missing X-Frame-Options Header

YES i

Missing HSTS header

YES i

Missing X-Content-Type-Options Header

YES i

Missing Content Security Policy (CSP)

YES i

HTTP Access Allowed

NO ✓

Self-Signed Certificate

NO ✓

Wrong Host Certificate

NO ✓

Expired Certificate

NO ✓

SSL/TLS Supports Weak Cipher

NO ✓

Support SSL Protocols

NO ✓

Support TLS Weak Version

NO ✓



## DNS Health

✓ 10 Passed

i 0 Attention

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





# Social Media Checks

✓ 2 Passed

i 7 Failed

X (Twitter)



PASS ✓

Facebook

FAIL ✗

Instagram

FAIL ✗

TikTok

FAIL ✗

YouTube

FAIL ✗

Twich

FAIL ✗

Telegram



PASS ✓

Discord

FAIL ✗

Others

FAIL ✗

## Recommendation

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

## Social Media Information Notes

## Unspecified Auditor Notes

## Notes from the Project Owner



# Fundamental Health

## KYC Status

SphinxShield KYC

3rd Party KYC

**NO** 



## Project Maturity Metrics

Minimally Developed

**LOW**

Token Launch Date

**NOT AVAILABLE**

Token Market Cap

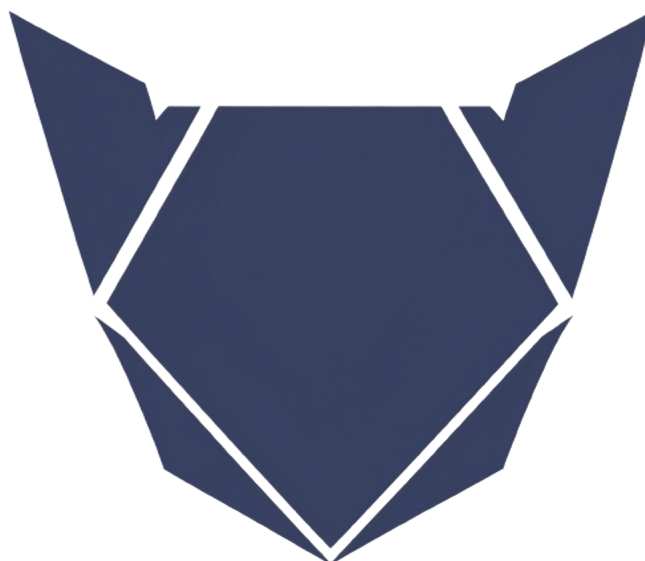
**NOT AVAILABLE**

Token/Project Age

**NOT AVAILABLE**

## Recommendation

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





# Disclaimer

SphinxShield, its agents, and associates provide the information and content contained within this audit report and materials (collectively referred to as "the Services") for informational and security assessment purposes only. The Services are provided "as is" without any warranty or representation of any kind, either express or implied. SphinxShield, its agents, and associates make no warranty or undertaking, and do not represent that the Services will:

- Meet customer's specific requirements.
- Achieve any intended results.
- Be compatible or work with any other software, applications, systems, or services.
- Operate without interruption.
- Meet any performance or reliability standards.
- Be error-free or that any errors or defects can or will be corrected.

In addition, SphinxShield, its agents, and associates make no representation or warranty of any kind, express or implied, as to the accuracy, reliability, or currency of any information or content provided through the Services. SphinxShield assumes no liability or responsibility for any:

- Errors, mistakes, or inaccuracies of content and materials.
- Loss or damage of any kind incurred as a result of the use of any content.
- Personal injury or property damage, of any nature whatsoever, resulting from customer's access to or use of the Services, assessment report, or other materials.

It is imperative to recognize that the Services, including any associated assessment reports or materials, should not be considered or relied upon as any form of financial, tax, legal, regulatory, or other advice.

SphinxShield provides the Services solely to the customer and for the purposes specifically identified in this agreement. The Services, assessment reports, and accompanying materials may not be relied upon by any other person or for any purpose not explicitly stated in this agreement. Copies of these materials may not be delivered to any other person without SphinxShield's prior written consent in each instance.

Furthermore, no third party or anyone acting on behalf of a third party shall be considered a third-party beneficiary of the Services, assessment reports, and any accompanying materials. No such third party shall have any rights of contribution against SphinxShield with respect to the Services, assessment reports, and any accompanying materials.

The representations and warranties of SphinxShield contained in this agreement are solely for the benefit of the customer. Accordingly, no third party or anyone acting on behalf of a third party shall be considered a third-party beneficiary of such representations and warranties. No such third party shall have any rights of contribution against SphinxShield with respect to such representations or warranties or any matter subject to or resulting in indemnification under this agreement or otherwise.



# About

SphinxShield, established in 2023, is a cybersecurity and auditing firm dedicated to fortifying blockchain and cryptocurrency security. We specialize in providing comprehensive security audits and solutions, aimed at protecting digital assets and fostering a secure investment environment.

Our accomplished team of experts possesses in-depth expertise in the blockchain space, ensuring our clients receive meticulous code audits, vulnerability assessments, and expert security advice. We employ the latest industry standards and innovative auditing techniques to reveal potential vulnerabilities, guaranteeing the protection of our clients' digital assets against emerging threats.

At SphinxShield, our unwavering mission is to promote transparency, security, and compliance with industry standards, contributing to the growth of blockchain and cryptocurrency projects. As a forward-thinking company, we remain adaptable, staying current with emerging trends and technologies to consistently enhance our services.

SphinxShield is your trusted partner for securing crypto ventures, empowering you to explore the vast potential of blockchain technology with confidence.

