

# SPHINXSHIELD

## Security Assessment

## Starship

## Oct 24th, 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	94/100
Auditor Score	92/100

Review by Section	Score
Manual Scan Score	49/57
Advance Check Score	17/19

## 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 **Starship**, aiming to uncover potential issues and vulnerabilities within the **Starship** 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	Starship
Blockchain	Ethereum
Language	Solidity
Codebase	<a href="https://etherscan.io/address/0xc1ecfaf43c53bec9b9143ab274f35603fd10b886">https://etherscan.io/address/0xc1ecfaf43c53bec9b9143ab274f35603fd10b886</a>
Commit	eadc1655a9a8abbf8e31f07fa8f1f182c41af11f3511b0d925cbd38597959071

## Audit Summary

Delivery Date	Oct 24th, 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	0	0	0	0	0
🟢 Low	1	0	0	1	0
🔵 Informational	8	0	0	8	0
🟢 Discussion	0	0	0	0	0

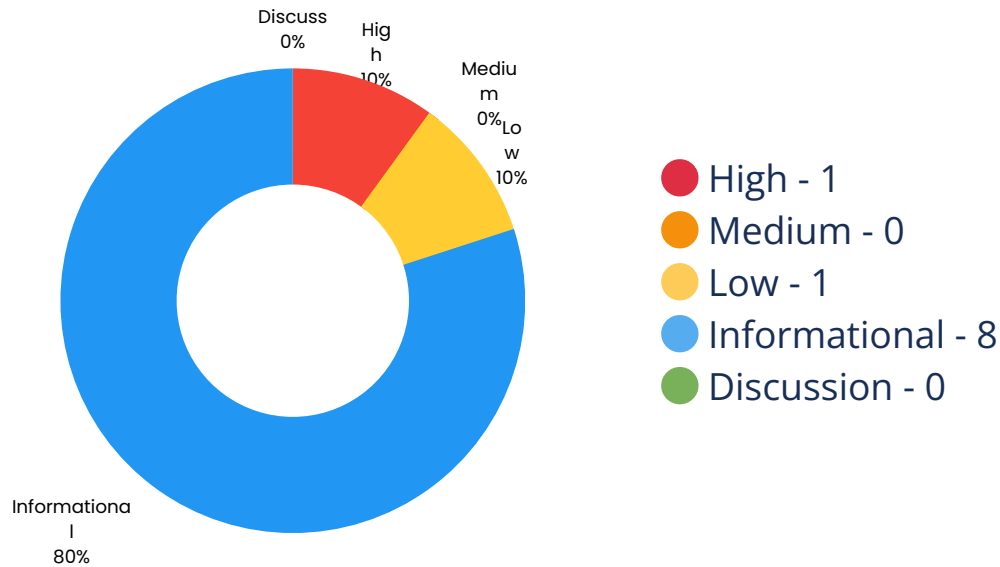


## Audit Scope

ID	File	KECCAK256 or SHA256 Checksum
SPM	StarShip.sol	0xb10f28971cdb1766a2c6cea4f7203a414e99d0db1e0f7f166dcf10d0416078a8



# Findings



Location	Title	Scope	Severity	Status
StarShip.sol:599	Freeze Money	StarShip	High	Aknowledged
StarShip.sol:528,527,529,498	Uninitialized Variables	StarShip	Low	Aknowledged
StarShip.sol:124,87,650,647,262,118,135,82,669,113,268,658,92,107,141,653	Function Visibility Can Be External	ERC20	Informational	Aknowledged
StarShip.sol:628,642,659,621,185,269,257,160,620,197,643	Use CustomError Instead of String	SafeMath	Informational	Aknowledged
StarShip.sol:6,477,232,14	Recommend to Follow Code Layout Conventions	IERC20	Informational	Aknowledged
StarShip.sol:628,621,269,620,197	Long String in revert/require	SafeMath	Informational	Aknowledged
StarShip.sol:124,658,664,650,107,647,118,135,141,653	No Check of Address Params with Zero Address	ERC20	Informational	Aknowledged



Location	Title	Scope	Severity	Status
StarShip.sol:269,160	Use Assembly to Check Zero Address	Ownable	● Informational	Aknowledged
StarShip.sol:477,63	No Need To Use SafeMath in Solidity Contract of Version 0.8.0 and Above	ERC20	● Informational	Aknowledged
StarShip.sol:554,553	Cache State Variables that are Read Multiple Times within A Function	StarShip	● Informational	Aknowledged





## Code Security – Freeze Money

Title	Severity	Location	Status
Freeze Money	● High	StarShip.sol:599	Aknowledged

### Description

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

## Code Security – Uninitialized Variables

Title	Severity	Location	Status
Uninitialized Variables	● Low	StarShip.sol:528,527,529,498	Aknowledged

### Description

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

## Optimization Suggestion – Function Visibility Can Be External

Title	Severity	Location	Status
Function Visibility Can Be External	● Informational	StarShip.sol:124,87,650,647,262,118,135,82,669,113,268,658,92,107,141,653	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	<span>●</span> Informational	StarShip.sol:628,642,6 59,621,185,269,257,16 0,620,197,643	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 – Recommend to Follow Code Layout Conventions

Title	Severity	Location	Status
Recommend to Follow Code Layout Conventions	<span>●</span> Informational	StarShip.sol:6,477,232 ,14	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 – Long String in revert/require

Title	Severity	Location	Status
Long String in revert/require	<span>●</span> Informational	StarShip.sol:628,621,2 69,620,197	Aknowledged

### Description

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



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

Title	Severity	Location	Status
No Check of Address Params with Zero Address	<span>●</span> Informational	StarShip.sol:124,658,6 64,650,107,647,118,13 5,141,653	Aknowledged

### Description

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

## Optimization Suggestion – Use Assembly to Check Zero Address

Title	Severity	Location	Status
Use Assembly to Check Zero Address	<span>●</span> Informational	StarShip.sol:269,160	Aknowledged

### Description

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

## 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	StarShip.sol:477,63	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 – Cache State Variables that are Read Multiple Times within A Function

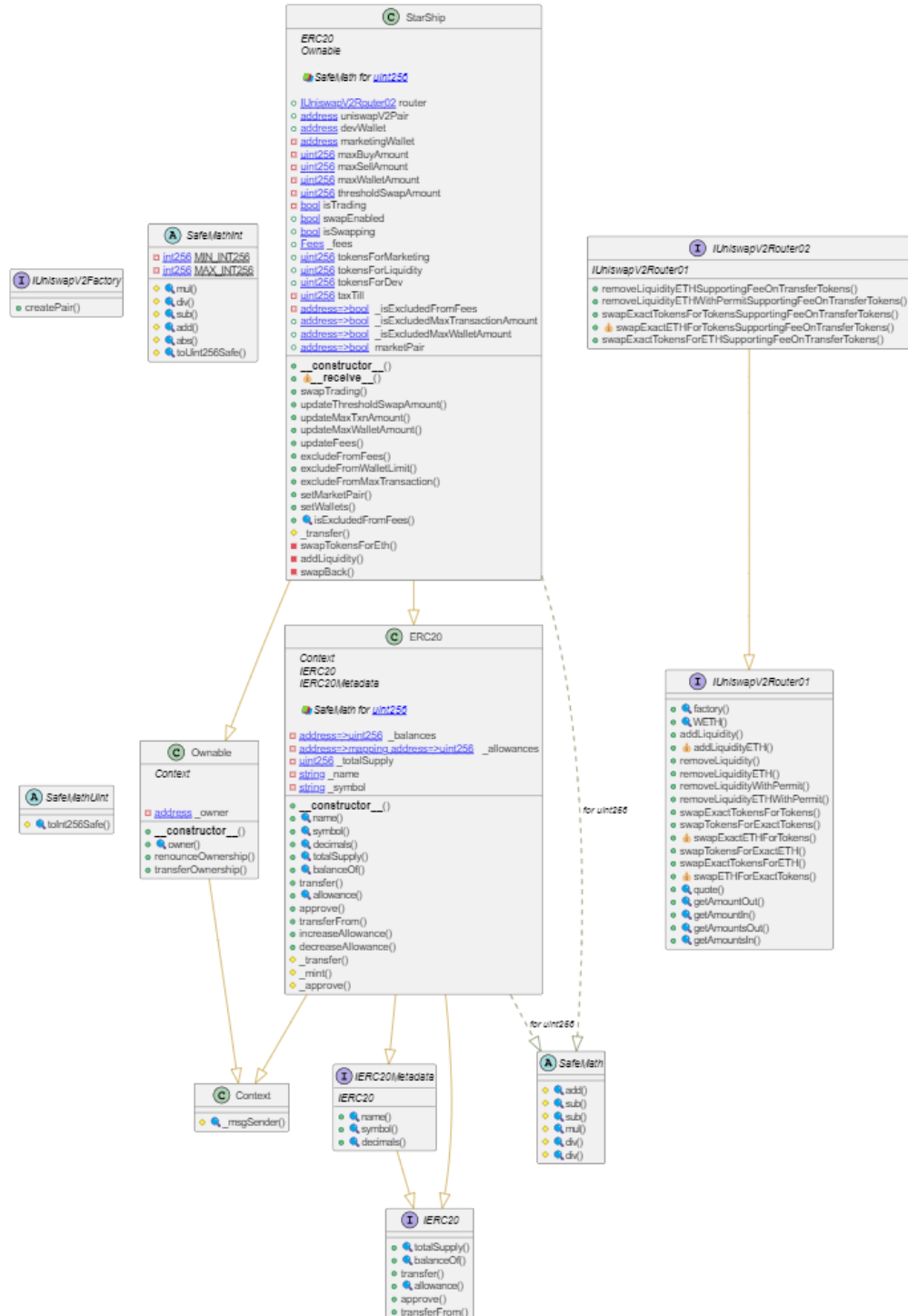
Title	Severity	Location	Status
Cache State Variables that are Read Multiple Times within A Function	<span>●</span> Informational	StarShip.sol:554,553	Acknowledged

### 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.



# PlantUML





# Appendix

## Finding Categories

### **Security and Best Practices**

1. Freeze Money: Implement mechanisms to freeze funds when necessary to protect the system and its users from potential threats.
2. Uninitialized Variables: Always initialize variables to prevent unexpected behavior and vulnerabilities.
3. Function Visibility Can Be External: Enhance gas efficiency by setting functions to external visibility if they are accessible only from within the contract.
4. Use CustomError Instead of String: Utilize custom error codes instead of string error messages for more efficient contract operation.
5. Recommend to Follow Code Layout Conventions: Strict adherence to established code layout conventions can significantly improve code readability and maintainability.
6. Long String in revert/require: Optimize long revert or require strings to minimize gas usage and improve efficiency.
7. No Check of Address Params with Zero Address: Ensure address parameters include checks to verify they are not the zero address.
8. Use Assembly to Check Zero Address: Implement optimized assembly checks to verify zero addresses efficiently.
9. No Need To Use SafeMath in Solidity Contract of Version 0.8.0 and Above: Solidity versions 0.8.0 and above feature built-in overflow and underflow protection, reducing the need for SafeMath library usage.
10. Cache State Variables that are Read Multiple Times within A Function: Optimize contract efficiency by caching state variables read multiple times within a function.



## 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://www.starshiperc20.com/>



## Network Security

High | 0 Attentions

## Application Security

High | 4 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

✓ 9 Passed

i 2 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

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

**NO** 

3rd Party KYC

**NO** 

## Project Maturity Metrics

Minimally Developed

**LOW**

Token Launch Date

**2023.10.24 14:30 (UTC)**

Token Market Cap (estimate)

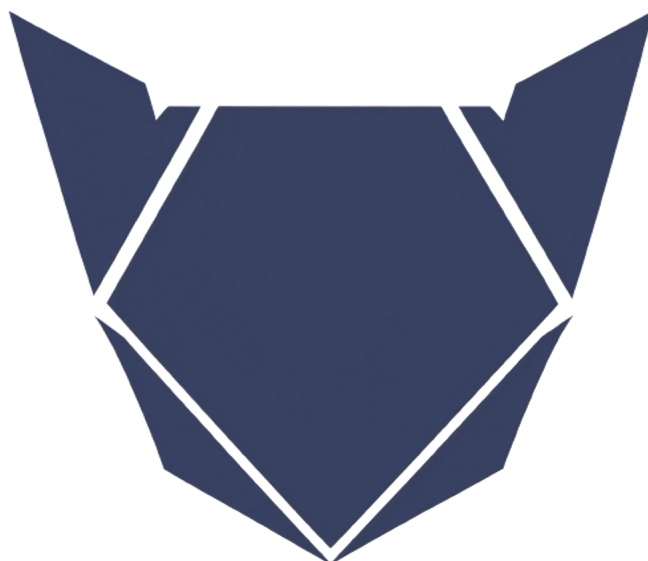
**\$13,021**

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.

