

# SPHINXSHIELD

## Security Assessment

## **Baby Dragon**

## Jan 11th, 2024

Disclaimer: SphinxShield conducts security assessments on the provided source code exclusively.  
Conduct your own due diligence before deciding to use any info listed at this page.



# Evaluation Outcomes

## Security Score

Review	Score
Overall Score	89/100
Auditor Score	83/100

Review by Section	Score
Manual Scan Score	46/57
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](#)

### **Understandings**

### **Findings**

### **PlantUML**

### **Appendix**

### **Website Scan**

### **Social Media Checks**

### **Fundamental Health**

### **Coin Tracker Analytics**

### **CEX Holding Analytics**

### **Disclaimer**

### **About**



# Summary

This audit report is tailored for **Baby Dragon**, aiming to uncover potential issues and vulnerabilities within the **Baby Dragon** 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	Baby Dragon
Blockchain	Binance Smart Chain
Language	Solidity
Codebase	<a href="https://bscscan.com/token/0xd7935bF3a576E997021789F895F0E898fC1aADd6">https://bscscan.com/token/0xd7935bF3a576E997021789F895F0E898fC1aADd6</a>
Commit	18be2551f8fdbcb1991096b3e47895c3766c088c9ea29c0ce67611788d52ac85b

## Audit Summary

Delivery Date	Jan 11th, 2024
Audit Methodology	Static Analysis, Manual Review
Key Components	BabyDragon.sol

## Vulnerability Summary



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



## Audit Scope

ID	File	KECCAK256 or SHA256 Checksum
BBD	BabyDragon.sol	0x8524e2115dab7f983fdc2be222653b2d5e9fae1122fe4accbe92a8eec3403329



# Understandings

BabyDragon is a decentralized finance (DeFi) token operating on the Binance Smart Chain (BSC). Deployed as the BabyDragon contract, this token introduces a reward system for creative contributions. Below is an overview of key features and functionalities within the BabyDragon contract:

## Token Information

- Token Name: Baby Dragon
- Symbol: BabyDragon
- Decimals: 9
- Total Supply: 420,000,000,000 BabyDragon

## Fee Management

The contract provides flexibility in managing various fees:

- Set Tax: The owner can configure liquidity, team, marketing, dev, and burn fees, along with the fee denominator, allowing adjustments to 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: Addresses that receive fee components (auto-liquidity, marketing, dev, and team fees) can be set by the owner.

## Tax Distribution

Transactions in BabyDragon incur a total fee, divided into components:

- Liquidity Fee: Collected for providing liquidity to the token, with a value set by the owner.
- Team Fee: A portion allocated to the project's team, adjustable by the owner.
- Marketing Fee: Part of the fee allocated to marketing efforts, configurable by the owner.
- Dev Fee: Allocated for development purposes, with the value set by the owner.
- Burn Fee: Tokens are burned, reducing the total supply. The burn fee is adjustable by the owner.
- Total Fee: The sum of the fees collected in each transaction, used to calculate overall fee distribution.
- Fee Denominator: The denominator used in fee calculations, usually set to 1,000.



## **Ownership and Authorization**

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

## **Transaction Limits**

The contract enforces transaction limits to prevent excessive token movement, ensuring users do not exceed defined limits.

## **Swap Mechanism**

BabyDragon employs a swap mechanism to manage liquidity. When a set threshold of tokens is reached, a portion of the contract's balance is swapped to BNB via the PancakeSwap Router. This swap action may temporarily affect the token's price, with the remaining balance supplied to the BabyDragon-BNB liquidity pool.

## **Open Trading**

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

## **Additional Functionality**

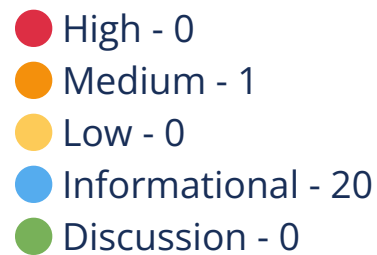
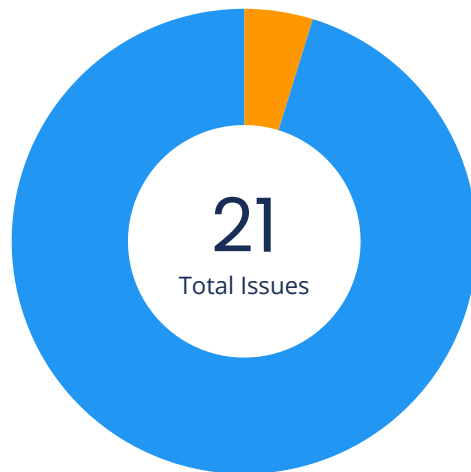
The contract includes other functions, such as clearing stuck ETH, clearing tokens, and more.

This understanding provides insights into the key features and functions of the BabyDragon contract deployed on the Binance Smart Chain. It serves as a vital component of the BabyDragon project's infrastructure, governing various aspects of its operation, including fees, liquidity, and user interactions.





# Findings



Location	Title	Scope	Severity	Status
BabyDragon.sol:405	Unchecked Call Return Value	BabyDragon	Medium	Acknowledged
BabyDragon.sol:179,180,181,182,183,184,193,195	Set the Constant to Private	BabyDragon	Informational	Acknowledged
BabyDragon.sol:25,134,158	Recommend to Follow Code Layout Conventions	Ownable	Informational	Acknowledged
BabyDragon.sol:60,145,146,214,216,217	Unused Events	IFactoryV2	Informational	Acknowledged
BabyDragon.sol:409	No Check of Address Params with Zero Address	BabyDragon	Informational	Acknowledged
BabyDragon.sol:268,333	Unused Internal Function	BabyDragon	Informational	Acknowledged
BabyDragon.sol:160	Redundant Getter Function	BabyDragon	Informational	Acknowledged
BabyDragon.sol:272	Continuous State Variable Write	BabyDragon	Informational	Acknowledged



Location	Title	Scope	Severity	Status
BabyDragon.sol:24 0,250,276,306,347, 390	Cache State Variables that are Read Multiple Times within A Function	BabyDragon	● Informational	Acknowledged
BabyDragon.sol:33 7,426	Use != 0 Instead of > 0 for Unsigned Integer Comparison	BabyDragon	● Informational	Acknowledged
BabyDragon.sol:28 6	Function Visibility Can Be External	BabyDragon	● Informational	Acknowledged
BabyDragon.sol:7	Floating Pragma	Global	● Informational	Acknowledged
BabyDragon.sol:39, 48,160,161,236,269 ,270,287,322,328,3 35,336,337,340,341 ,363,410,418,425,4 26,433	Use CustomError Instead of String	Ownable	● Informational	Acknowledged
BabyDragon.sol:16 0,161	Lack of Error Message	BabyDragon	● Informational	Acknowledged
BabyDragon.sol:38 4	ReentrancyGuard Should Modify External Function	BabyDragon	● Informational	Acknowledged
BabyDragon.sol:48, 335,336,337	Long String in revert/require	Ownable	● Informational	Acknowledged
BabyDragon.sol:19 0,196,200,201	Variables Can Be Declared as Immutable	BabyDragon	● Informational	Acknowledged
BabyDragon.sol:29 1,300,310,315,369, 384	Internal Functions Only Called Once Can Be Inlined	BabyDragon	● Informational	Acknowledged
BabyDragon.sol:40 5	Get Contract Balance of ETH in Assembly	BabyDragon	● Informational	Acknowledged
BabyDragon.sol:48, 269,270,287,296,32 2,335,336,363	Use Assembly to Check Zero Address	Ownable	● Informational	Acknowledged
BabyDragon.sol:26 3	ERC20   Missing Events Emitting in approve()	BabyDragon	● Informational	Acknowledged



## Code Security – Unchecked Call Return Value

Title	Severity	Location	Status
Unchecked Call Return Value	● Medium	BabyDragon.sol:405	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.

## Optimization Suggestion – Set the Constant to Private

Title	Severity	Location	Status
Set the Constant to Private	● Informational	BabyDragon.sol:179,180,181,182,183,184,193,195	Aknowledged

### Description

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

## Optimization Suggestion – Recommend to Follow Code Layout Conventions

Title	Severity	Location	Status
Recommend to Follow Code Layout Conventions	● Informational	BabyDragon.sol:25,134,158	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	<span>●</span> Informational	BabyDragon.sol:60,145,146,214,216,217	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	<span>●</span> Informational	BabyDragon.sol:409	Aknowledged
----------------------------------------------	------------------------------	--------------------	-------------

### Description

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

## Optimization Suggestion – Unused Internal Function

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

Unused Internal Function	<span>●</span> Informational	BabyDragon.sol:268,333	Aknowledged
--------------------------	------------------------------	------------------------	-------------

### Description

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



## Optimization Suggestion – Redundant Getter Function

Title	Severity	Location	Status
Redundant Getter Function	<span>●</span> Informational	BabyDragon.sol:160	Aknowledged

### Description

A state variable with public visibility comes with a getter function to obtain the value of the variable, so there is no need to explicitly define a function to obtain the value of the variable. When deploying a contract, remove extra functions will save gas. About 37000 gas can be saved with optimization turned off while 6800 gas can be saved vice versa.

## Optimization Suggestion – Continuous State Variable Write

Title	Severity	Location	Status
Continuous State Variable Write	<span>●</span> Informational	BabyDragon.sol:272	Aknowledged

### Description

When there are multiple continuous write operations on a state variable, the intermediate write operations are redundant and will cost more gas.

## Optimization Suggestion – 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	BabyDragon.sol:240,250,276,306,347,390	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 – Use != 0 Instead of > 0 for Unsigned Integer Comparison

Title	Severity	Location	Status
Use != 0 Instead of > 0 for Unsigned Integer Comparison	<span>●</span> Informational	BabyDragon.sol:337,426	Acknowledged

### 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	<span>●</span> Informational	BabyDragon.sol:286	Acknowledged

### Description

Functions that are not called should be declared as external.

## Optimization Suggestion – Floating Pragma

Title	Severity	Location	Status
Floating Pragma	<span>●</span> Informational	BabyDragon.sol:7	Acknowledged

### Description

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



## Optimization Suggestion – Use CustomError Instead of String

Title	Severity	Location	Status
Use CustomError Instead of String	<span>●</span> Informational	BabyDragon.sol:39,48,160,161,236,269,270,287,322,328,335,336,337,340,341,363,410,418,425,426,433	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	<span>●</span> Informational	BabyDragon.sol:160,161	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	<span>●</span> Informational	BabyDragon.sol:384	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

BabyDragon.sol:48,33  
5,336,337

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

BabyDragon.sol:190,1  
96,200,201

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 – Internal Functions Only Called Once Can Be Inlined

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

Internal Functions Only Called Once Can Be Inlined

● Informational

BabyDragon.sol:291,3  
00,310,315,369,384

Aknowledged

### Description

Inlining internal functions that are only called once into the external function can save gas. When compiler optimization is turned off, deploying the contract can save approximately 3000 gas, and calling the function can save approximately 40 gas. When compiler optimization is turned on, deploying the contract can save approximately 2000 gas, and calling the function can save approximately 50 gas.





## Optimization Suggestion - Get Contract Balance of ETH in Assembly

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

Get Contract Balance of ETH in Assembly	● Informational	BabyDragon.sol:405	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	BabyDragon.sol:48,26 9,270,287,296,322,335 ,336,363	Aknowledged
------------------------------------	-----------------	-----------------------------------------------------------	-------------

### Description

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

## Optimization Suggestion - ERC20 | Missing Events Emitting in approve()

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

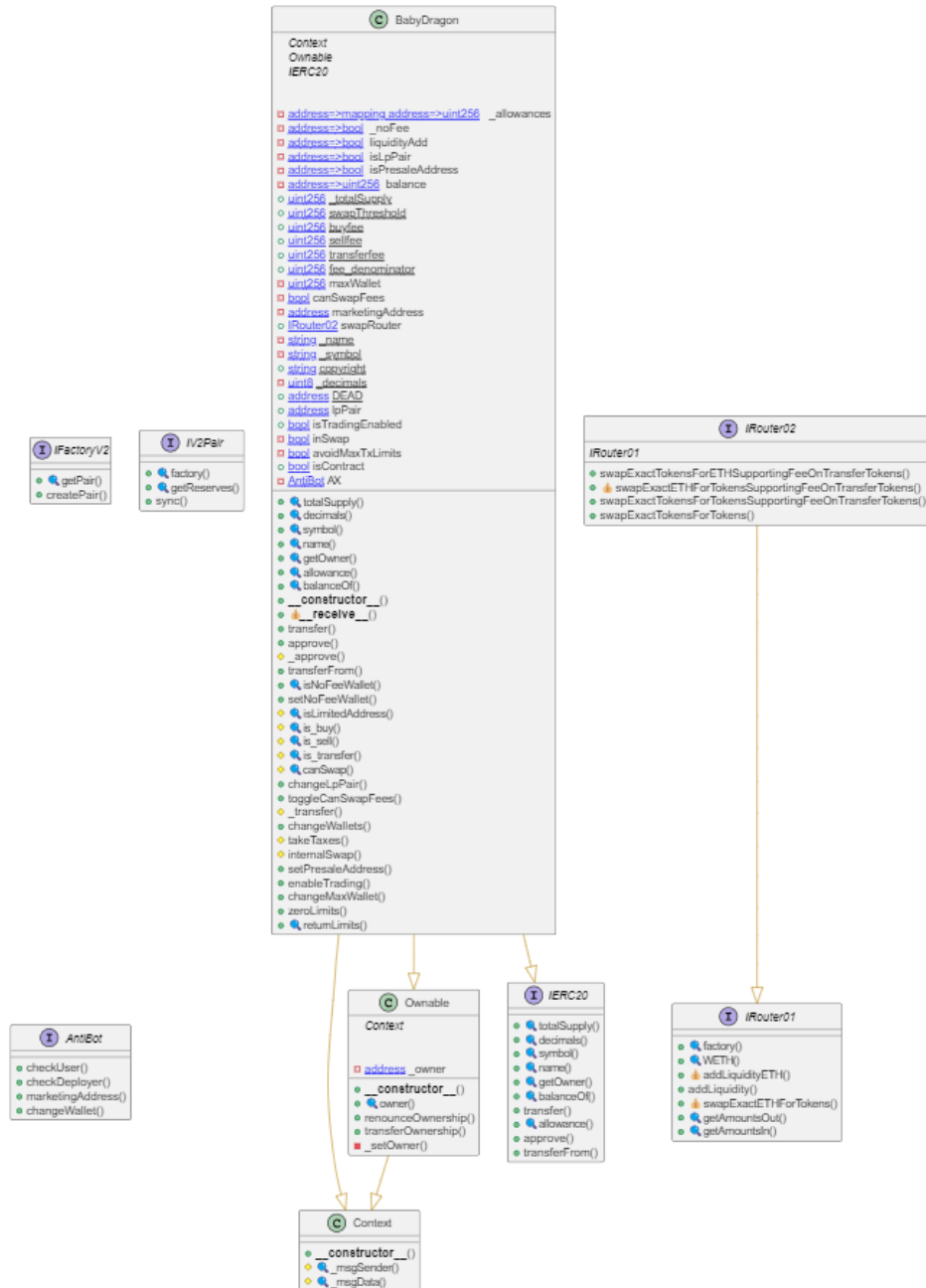
ERC20   Missing Events Emitting in approve()	● Informational	BabyDragon.sol:263	Aknowledged
----------------------------------------------	-----------------	--------------------	-------------

### Description

ERC20 - The approve function do not emit Approval event.



# PlantUML





# Appendix

## Finding Categories

### Security and Best Practices

1. **Unchecked Call Return Value:** Exercise caution when utilizing external calls without validating their return values. Failing to check the return value can lead to unexpected behavior and potential vulnerabilities.
2. **Set the Constant to Private:** Constants should be designated as private to restrict external access and modification. This practice enhances contract security by preventing unintended manipulation.
3. **Recommend to Follow Code Layout Conventions:** Adhering to established code layout conventions promotes readability and maintainability. Consistent formatting conventions make the codebase more accessible to developers.
4. **Unused Events:** Unused events may indicate redundant or incomplete code. Remove or utilize events effectively to ensure comprehensive logging and facilitate easier debugging.
5. **No Check of Address Params with Zero Address:** Verify address parameters to prevent potential vulnerabilities associated with the zero address. Ensure that functions handling addresses include appropriate checks.
6. **Unused Internal Function:** Unused internal functions contribute to code bloat and can be a source of confusion. Remove or repurpose these functions to streamline the contract.
7. **Redundant Getter Function:** Avoid unnecessary getter functions that duplicate existing functionality. Redundant getters can clutter the contract and impact gas efficiency.
8. **Continuous State Variable Write:** Minimize continuous state variable writes within a function to enhance gas efficiency. Limiting state modifications can lead to more cost-effective contract execution.
9. **Cache State Variables that are Read Multiple Times within A Function:** Optimize gas consumption by caching state variables that are read multiple times within a function. Reducing redundant reads can result in more efficient contract execution.
10. **Use != 0 Instead of > 0 for Unsigned Integer Comparison:** Prefer using != 0 over > 0 for unsigned integer comparisons. This practice ensures consistency and reduces the likelihood of unintended behavior.
11. **Function Visibility Can Be External:** Enhance gas efficiency by setting functions to external visibility if they are intended to be accessed only from outside the contract.
12. **Floating Pragma:** Maintain a consistent Solidity pragma version for added contract security. Avoid using floating pragmas to ensure predictable compilation.
13. **Use CustomError Instead of String:** Opt for custom error codes instead of string error messages for more efficient contract operation. This minimizes gas consumption and simplifies error handling.
14. **Lack of Error Message:** Include informative error messages to aid in debugging and improve user experience. Providing clear error messages enhances the contract's usability.
15. **ReentrancyGuard Should Modify External Function:** Ensure that the ReentrancyGuard is appropriately applied to external functions to prevent reentrancy attacks. This practice enhances the contract's security.
16. **Long String in revert/require:** Long revert or require strings can increase gas usage. Optimize these strings for gas efficiency to minimize transaction costs.
17. **Variables Can Be Declared as Immutable:** Declare variables as immutable if they do not change after initialization. This practice enhances security and readability by explicitly conveying the variable's intended immutability.
18. **Internal Functions Only Called Once Can Be Inlined:** Consider inlining internal functions that are called only once. This optimization can lead to more streamlined and readable code.
19. **Get Contract Balance of ETH in Assembly:** Optimize gas efficiency by using assembly to retrieve the contract's ETH balance. Assembly allows for more efficient and concise operations.
20. **Use Assembly to Check Zero Address:** Leverage optimized assembly checks to verify zero addresses efficiently. Assembly provides a more gas-efficient solution for this specific validation.
21. **ERC20 | Missing Events Emitting in approve():** Ensure comprehensive event logging in the approve() function to provide transparency and facilitate effective monitoring of token approvals in ERC-20 contracts.



## 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://baby-dragon.vip/>



## Network Security

**High** | 0 Attentions

## Application Security

**Med** | 5 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



6 Passed



5 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**

NO



## DNS Health

✓ 7 Passed

i 3 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**

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

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

Minimally Developed

**LOW**

Token Launch Date

**2024.01.07 14:00 (UTC)**

Token Market Cap (estimate)

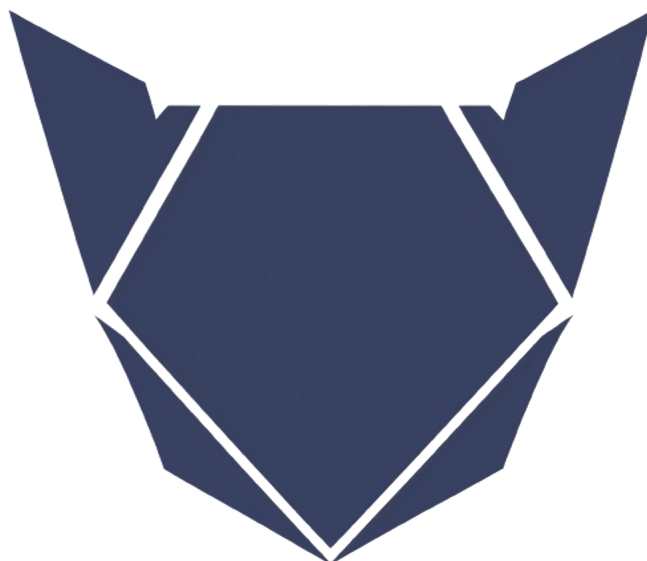
**\$1.30M**

Token/Project Age

**7 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

NO 



CoinGecko

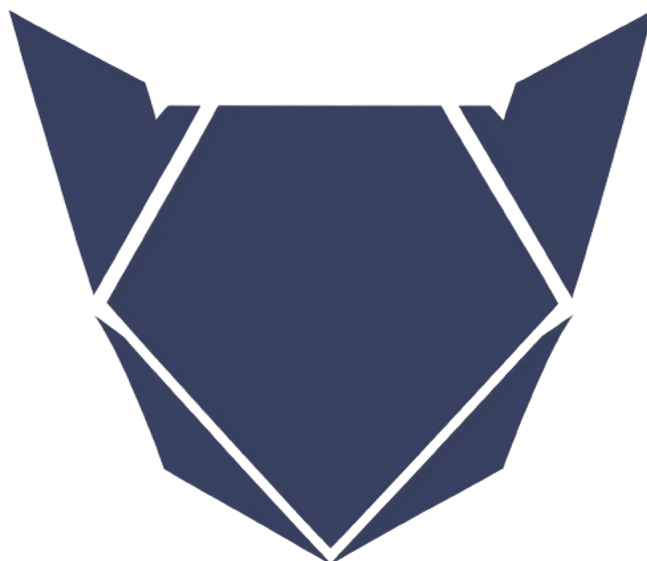
YES 

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.

