SPHINXSHIELD

Security Assessment

Priv2Fans

Jan 4th, 2024

Evaluation Outcomes

Security Score

Review	Score
Overall Score	89/100
Auditor Score	87/100

Review by Section	Score
Manual Scan Score	47/57
Advance Check Score	16/19

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

This audit report is tailored for **Priv2Fans**, aiming to uncover potential issues and vulnerabilities within the **Priv2Fans** 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

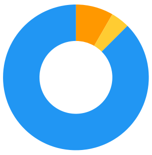
Project Summary

Project Name	Priv2Fans
Blockchain	Binance Smart Chain
Language	Solidity
Codebase	https://bscscan.com/address/0x59931E9655e0dcE109C5A4dAF66B1e9D85d38125
Commit	6b3cdcd747ccffde0f872c7b21c646b780b2fab45f8bd8c6f4b73570b4a54b65

Audit Summary

Delivery Date	Jan 4th, 2024
Audit Methodology	Static Analysis, Manual Review
Key Components	Priv2Fans.sol

Vulnerability Summary



24	0	0	0	24	0
Total Findings	Resolved	Mitigated	Partially Resolved	Aknowledged	Declined

Vulnerability Level	Total	Pending	Declined	Aknowledged	Resolved
High	0	0	0	0	0
Medium	2	0	0	2	0
Low	1	0	0	1	0
Informational	21	0	0	21	0
Discussion	0	0	0	0	0

Audit Scope

ID	File	KECCAK256 or SHA256 Checksum
FAN	Priv2Fans.sol	

Understandings

Priv2Fans is a decentralized finance (DeFi) token deployed on the Binance Smart Chain (BSC) blockchain. The contract, named LiquidityGeneratorToken, incorporates various features and mechanisms to manage its operations, including tax distribution, liquidity acquisition, and privileged functions for modifying contract parameters. Here's an in-depth breakdown of the key components and functionalities within the Priv2Fans contract:

Token Information

- Token Name: Priv2Fans
- Symbol: FANS
- Decimals: 9
- Total Supply: 1,000,000,000,000,000,000 FANS

Fee Structure

Priv2Fans transactions incur a total fee, which is divided into various components:

- Tax Fee: This fee is calculated based on the specified tax fee percentage (taxFeeBps) and is used for various purposes, including community rewards and development.
- Liquidity Fee: Collected for providing liquidity to the token. Its value is set by the owner.
- Charity Fee: A portion of the fee is allocated to charitable contributions. The charity fee is adjustable by the owner.
- Total Fee: The sum of tax, liquidity, and charity fees.

Fee Management

The contract allows the owner to manage various fees:

- Set Tax Fee: The owner can configure the tax fee, liquidity fee, and charity fee, providing flexibility in managing the fee structure.
- Set Fee Multipliers: The owner can set multipliers to adjust the percentage of fees for buy, sell, and transfer transactions.
- Set Fee Receivers: Addresses that receive various fee components (auto-liquidity, charity, etc.) can be set by the owner.

Tax Exemption

The contract provides the owner with the ability to exempt specific addresses from fees, offering a mechanism for fee exemption to certain addresses. This feature can be used for whitelisting specific wallets or contracts.

Ownership and Authorization

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

Transaction Limits

To prevent excessive token movement, the contract enforces transaction limits, ensuring that users do not make transactions that exceed the defined limits.

Swap Mechanism

Priv2Fans 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 (Binance Coin) via the PancakeSwap Router. This swap action temporarily affects the token's price. The remaining balance is then supplied to the Priv2Fans-BNB liquidity pool.

Open Trading

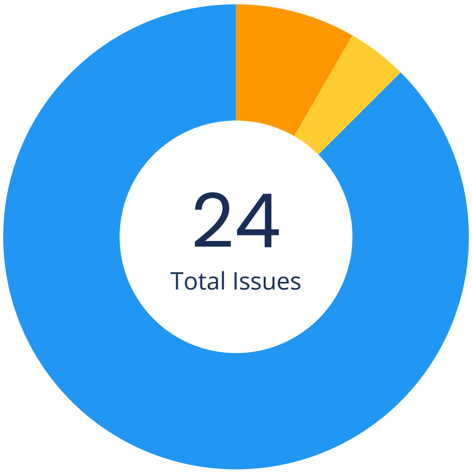
Trading can be restricted based on conditions defined by the owner. This ensures that trading remains closed until specific requirements are met.

Additional Functionality

The contract includes various additional functions, such as clearing stuck ETH, clearing tokens, and more, providing a comprehensive infrastructure for the Priv2Fans project.

Please note that the information provided here is based on the analysis of the Priv2Fans contract on the Binance Smart Chain.

Findings



High - 0
Medium - 2
Low - 1
Informational - 21
Discussion - 0

Location	Title	Scope	Severity	Status
Priv2Fans.sol:974	Unauthenticated Storage Access	LiquidityGeneratorToken	Medium	Aknowledged
Priv2Fans.sol:1065	Unauthenticated Storage Access	LiquidityGeneratorToken	Medium	Aknowledged
Priv2Fans.sol:1041	Use Safer Functions	LiquidityGeneratorToken	Low	Aknowledged
Priv2Fans.sol:1041	Prefer .call() To send()/transfer()	LiquidityGeneratorToken	Informational	Aknowledged
Priv2Fans.sol:940	Prefer uint256	LiquidityGeneratorToken	Informational	Aknowledged
Priv2Fans.sol:921,923	Set the Constant to Private	LiquidityGeneratorToken	Informational	Aknowledged
Priv2Fans.sol:10,131,917	Recommend to Follow Code Layout Conventions	IERC20	Informational	Aknowledged

Location	Title	Scope	Severity	Status
Priv2Fans.sol:850,896,960	Unused Events	IUniswapV2Factory	Informational	Aknowledged
Priv2Fans.sol:974,1186,1196,1233	No Check of Address Params with Zero Address	LiquidityGeneratorToken	Informational	Aknowledged
Priv2Fans.sol:917	No Need To Use SafeMath in Solidity Contract of Version 0.8.0 and Above	LiquidityGeneratorToken	Informational	Aknowledged
Priv2Fans.sol:907	Inconsistent Solidity Version	Global	Informational	Aknowledged
Priv2Fans.sol:923	Use Shift Operation Instead of Mul/Div	LiquidityGeneratorToken	Informational	Aknowledged
Priv2Fans.sol:1452	Continuous State Variable Write	LiquidityGeneratorToken	Informational	Aknowledged
Priv2Fans.sol:1198,1370	Use ++i/--i Instead of i++/i--	LiquidityGeneratorToken	Informational	Aknowledged
Priv2Fans.sol:1002,1189,1368,1369,1399,1404,1412,1420,1421,1472,1561	Cache State Variables that are Read Multiple Times within A Function	LiquidityGeneratorToken	Informational	Aknowledged
Priv2Fans.sol:1463	Use != 0 Instead of > 0 for Unsigned Integer Comparison	LiquidityGeneratorToken	Informational	Aknowledged
Priv2Fans.sol:1109,1122,1138,1142,1146,1158,1186,1233,1440	Function Visibility Can Be External	LiquidityGeneratorToken	Informational	Aknowledged
Priv2Fans.sol:907	Floating Pragma	Global	Informational	Aknowledged
Priv2Fans.sol:154,174,359,987,992,1148,1163,1178,1188,1197,1239,1250,1258,1265,1449,1450,1461,1462,1463	Use CustomError Instead of String	Ownable	Informational	Aknowledged

Location	Title	Scope	Severity	Status
Priv2Fans.sol:1496	ReentrancyGuard Should Modify External Function	LiquidityGeneratorToken	Informational	Aknowledged
Priv2Fans.sol:174,359,987,1148,1178,1265,1449,1450,1461,1462,1463	Long String in revert/require	Ownable	Informational	Aknowledged
Priv2Fans.sol:934,940,951,952,953,956	Variables Can Be Declared as Immutable	LiquidityGeneratorToken	Informational	Aknowledged
Priv2Fans.sol:1505	Get Contract Balance of ETH in Assembly	LiquidityGeneratorToken	Informational	Aknowledged
Priv2Fans.sol:174,986,1420,1449,1450,1461,1462	Use Assembly to Check Zero Address	Ownable	Informational	Aknowledged

Code Security - Unauthenticated Storage Access

Title	Severity	Location	Status
Unauthenticated Storage Access	Medium	Priv2Fans.sol:974	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	Priv2Fans.sol:1065	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	Priv2Fans.sol:1041	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	Priv2Fans.sol:1041	Aknowledged

Description

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

Optimization Suggestion – Prefer uint256

Title	Severity	Location	Status
Prefer uint256	Informational	Priv2Fans.sol:940	Aknowledged

Description

It is recommended to use uint256/int256 types to avoid gas overhead caused by 32 bytes padding.

Optimization Suggestion – Set the Constant to Private

Title	Severity	Location	Status
Set the Constant to Private	Informational	Priv2Fans.sol:921,923	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	Priv2Fans.sol:10,131,917	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	Priv2Fans.sol:850,896,960	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	Priv2Fans.sol:974,1186,1196,1233	Aknowledged

Description

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

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	Priv2Fans.sol:917	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 - Inconsistent Solidity Version

Title	Severity	Location	Status
Inconsistent Solidity Version	Informational	Priv2Fans.sol:907	Aknowledged

Description

The source files have different solidity compiler ranges referenced. This leads to potential security flaws between deployed contracts depending on the compiler version chosen for any particular file. It also increases the cost of maintenance as different compiler versions have different semantics and behavior.

Optimization Suggestion - Use Shift Operation Instead of Mul/Div

Title	Severity	Location	Status
Use Shift Operation Instead of Mul/Div	Informational	Priv2Fans.sol:923	Aknowledged

Description

It is recommended to use shift operation instead of direct multiplication and division if possible, because shift operation is more gas-efficient.

Optimization Suggestion - Continuous State Variable Write

Title	Severity	Location	Status
Continuous State Variable Write	Informational	Priv2Fans.sol:1452	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 - Use ++i/--i Instead of i++/i--

Title	Severity	Location	Status
Use ++i/--i Instead of i++/i--	Informational	Priv2Fans.sol:1198,1370	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	Priv2Fans.sol:1002,1189,1368,1369,1399,1404,1412,1420,1421,1472,1561	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	Informational	Priv2Fans.sol:1463	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	Priv2Fans.sol:1109,1122,1138,1142,1146,1158,1186,1233,1440	Aknowledged

Description

Functions that are not called should be declared as external.

Optimization Suggestion - Floating Pragma

Title	Severity	Location	Status
Floating Pragma	Informational	Priv2Fans.sol:154,174,359,987,992,1148,1163,1178,1188,1197,1239,1250,1258,1265,1449,1450,1461,1462,1463	Aknowledged

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 - ReentrancyGuard Should Modify External Function

Title	Severity	Location	Status
ReentrancyGuard Should Modify External Function	Informational	Priv2Fans.sol:1496	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	Priv2Fans.sol:174,359,987,1148,1178,1265,1449,1450,1461,1462,1463	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	Priv2Fans.sol:934,940,951,952,953,956	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	Priv2Fans.sol:1505	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	Priv2Fans.sol:174,986,1420,1449,1450,1461,1462	Aknowledged

Description

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

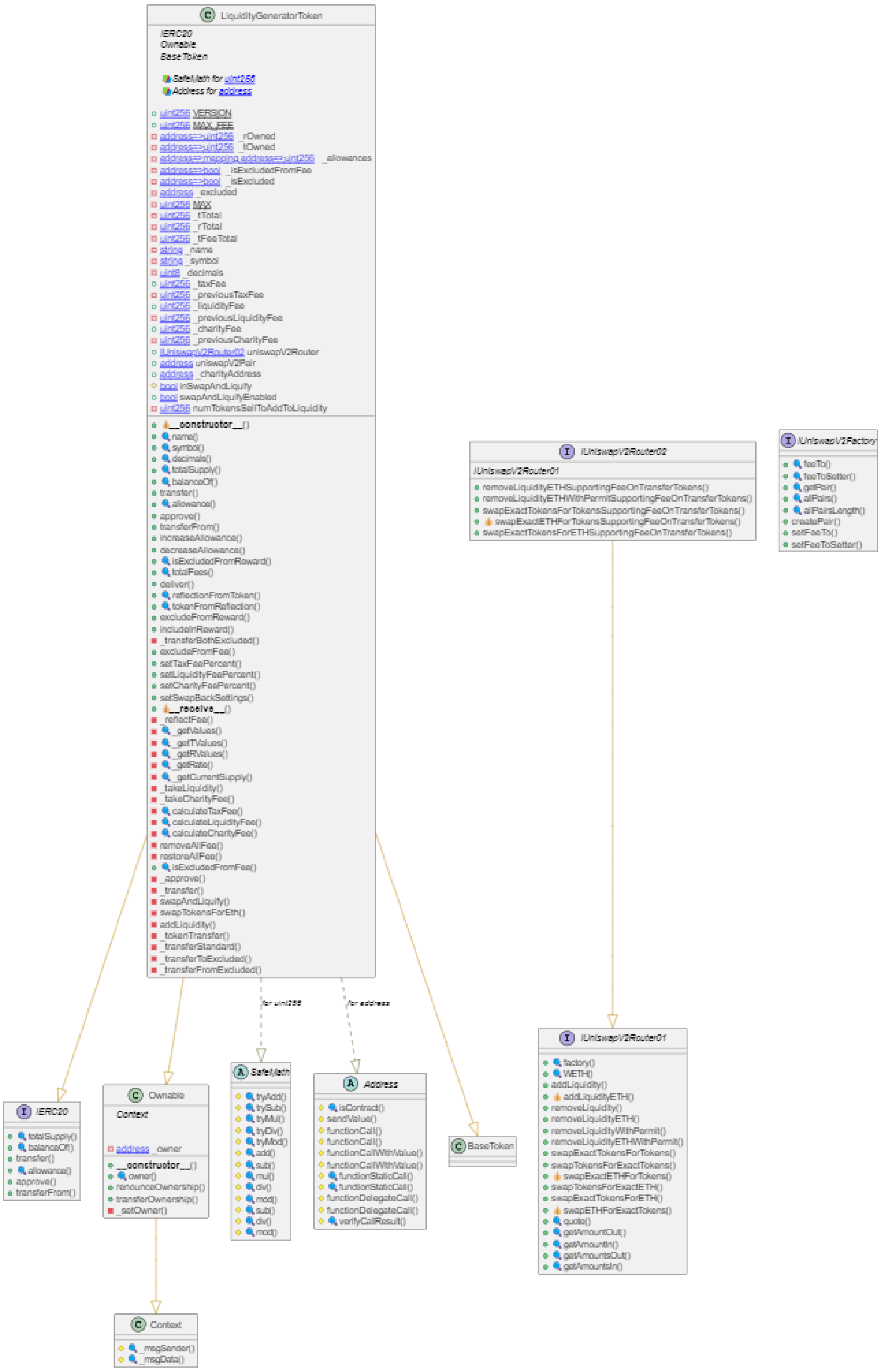
Optimization Suggestion - Use CustomError Instead of String

Title	Severity	Location	Status
Use CustomError Instead of String	Informational	Priv2Fans.sol:154,174,359,987,992,1148,1163,1178,1188,1197,1239,1250,1258,1265,1449,1450,1461,1462,1463	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.

PlantUML



LiquidityGeneratorToken
IERC20
Ownable
BaseToken
IUniswapV2Router02
IUniswapV2Router01
IUniswapV2Factory
for uint256
for address
IERC20
Ownable
Context
SafeMath
Address
BaseToken
IUniswapV2Router01
Context

Appendix

Finding Categories

Security and Best Practices

1. Unauthenticated Storage Access: Smart contracts should undergo scrutiny for unauthenticated storage access, which can lead to unauthorized data tampering. Implement robust access control mechanisms to prevent unauthorized modifications to contract storage.
2. Use Safer Functions: Utilize functions known for their secure design to mitigate potential security vulnerabilities. Review functions for enhanced security, favoring those with established safety records in the Ethereum community.
3. Prefer .call() To send()/transfer(): Employ .call() instead of send()/transfer() for external contract calls to minimize security risks. The use of .call() allows better handling of exceptions and provides more control over the execution flow.
4. Prefer uint256: Emphasize the use of uint256 over other data types to maintain consistency and enhance contract security. uint256 is the recommended data type for handling numerical values in Ethereum smart contracts.
5. Set the Constant to Private: Declared constants should be set to private visibility to prevent unwanted external access. This practice helps to encapsulate internal details and reduces the risk of unintended interference.
6. Recommend to Follow Code Layout Conventions: Strict adherence to established code layout conventions can significantly improve code readability and maintainability. Consistent formatting enhances collaboration among developers and facilitates code reviews.
7. Unused Events: Unused events in the contract should be removed to reduce unnecessary contract complexity and save gas costs.
8. No Check of Address Params with Zero Address: Verification of address parameters should include checks to ensure that the address is not the zero address. Failing to check for the zero address can lead to unexpected behavior in contract execution.
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, minimizing the necessity of SafeMath library usage. Review and remove redundant SafeMath implementations.
10. Inconsistent Solidity Version: Ensure consistency in the Solidity version used throughout the contract. Inconsistencies may lead to unexpected behaviors and should be avoided for a stable deployment.
11. Use Shift Operation Instead of Mul/Div: Employ shift operations (<< and >>) instead of multiplication and division where applicable. Shift operations are generally more gas-efficient for certain cases.
12. Continuous State Variable Write: Minimize continuous writes to state variables to reduce gas consumption. Frequent state variable writes can lead to higher transaction costs.
13. Use ++i/--i Instead of i++/i--: Prefer pre-increment (i++) and pre-decrement (i--) over post-increment and post-decrement to potentially optimize gas usage.
14. 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 state variable reads can lead to cost savings.
15. Use != 0 Instead of > 0 for Unsigned Integer Comparison: Utilize "!= 0" for checking if an unsigned integer is non-zero instead of "> 0" for improved clarity and consistency in code.
16. Function Visibility Can Be External: Enhance gas efficiency by setting functions to external visibility if they are accessible only from within the contract. External functions can be more cost-effective for certain use cases.
17. Floating Pragma: Ensure that your Solidity pragma remains consistent for added contract security. Avoid using floating pragmas that may unintentionally expose the contract to unforeseen risks.
18. Use CustomError Instead of String: Opt for custom error codes instead of string error messages for more efficient contract operation. String manipulation can be gas-intensive, and using custom error codes is a more gas-efficient alternative.

1. ReentrancyGuard Should Modify External Function: When using a ReentrancyGuard, ensure that it modifies external functions rather than internal ones to prevent reentrancy attacks.
2. Long String in revert/require: Long revert or require strings can increase gas usage and should be optimized for gas efficiency. Consider using concise error messages or custom error codes to minimize gas costs.
3. Variables Can Be Declared as Immutable: Variables that do not change after initialization can be declared as immutable to enhance security and readability. Immutable variables ensure that their values remain constant throughout the contract's execution.
4. Get Contract Balance of ETH in Assembly: Use assembly to efficiently retrieve the contract's ETH balance. This can be more gas-efficient than using the balance property in high-frequency scenarios.
5. Use Assembly to Check Zero Address: Optimized assembly checks can be employed to verify zero addresses efficiently. Leveraging assembly can lead to more gas-efficient and streamlined zero address checks.

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://priv2fans.com/

Network Security

High | 1 Attentions

Application Security

High | 4 Attentions

DNS Security

High | 6 Attentions

Network Security

8 Passed

1 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	YES

Application Security

7 Passed | 4 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	NO
Support SSL Protocols	NO
Support TLS Weak Version	NO

DNS Health

4 Passed

6 Attention

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

Social Media Checks

3 Passed | 7 Failed

X (Twitter)	PASS
Facebook	FAIL
Instagram	PASS
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

Emerging	**LOW**
Token Launch Date	**Not Launched**
Token Market Cap (estimate)	**Not Estimated**
Token/Project Age	**3 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	NO
Others	NO

Recommendation

We highly recommend that the project consider integrating with multiple coin tracking platforms to expand its visibility within the cryptocurrency ecosystem. In particular, joining prominent platforms such as CoinMarketCap and CoinGecko can significantly benefit the project by increasing its reach and credibility.

CEX Holding Analytics

Status

Not available on any centralized cryptocurrency exchanges (CEX).

Recommendation

To increase your project's visibility and liquidity, we recommend pursuing listings on centralized cryptocurrency exchanges. Here's a recommendation you can use:

We strongly advise the project team to actively pursue listings on reputable centralized cryptocurrency exchanges. Being listed on these platforms can offer numerous advantages, such as increased liquidity, exposure to a broader range of traders, and enhanced credibility within the crypto community.

To facilitate this process, we recommend the following steps:

1. Research and Identify Suitable Exchanges: Conduct thorough research to identify centralized exchanges that align with your project's goals and target audience. Consider factors such as trading volume, reputation, geographical reach, and compliance with regulatory requirements.
2. Meet Compliance Requirements: Ensure that your project is compliant with all necessary legal and regulatory requirements for listing on these exchanges. This may include Know Your Customer (KYC) verification, security audits, and legal documentation.
3. Prepare a Comprehensive Listing Proposal: Create a detailed and persuasive listing proposal for each exchange you intend to approach. This proposal should highlight the unique features and benefits of your project, as well as your commitment to compliance and security.
4. Engage in Communication: Establish open lines of communication with the exchange's listing team. Be prepared to address their questions, provide requested documentation, and work closely with their team to facilitate the listing process.
5. Marketing and Community Engagement: Promote your project within the exchange's community and among your own supporters to increase visibility and trading activity upon listing.
6. Maintain Transparency: Maintain transparency and provide regular updates to your community and potential investors about the progress of listing efforts.
7. Be Patient and Persistent: Listing processes on centralized exchanges can sometimes be lengthy. Be patient and persistent in your efforts, and consider seeking the assistance of experts or advisors with experience in exchange listings if necessary.
8.

Remember that listing on centralized exchanges can significantly impact your project's growth and market accessibility. By following these steps and maintaining a professional, compliant, and communicative approach, you can increase your chances of successfully getting listed on centralized exchanges.

Disclaimer

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

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

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

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

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

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

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

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

About

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

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

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

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