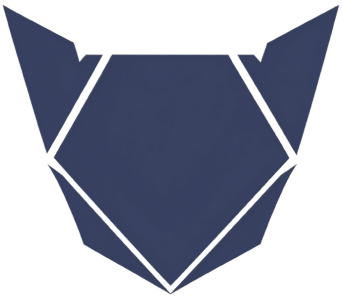


SPHINXSHIELD

# Security Assessment

# **Banger**

Nov 11th, 2023

# Evaluation Outcomes

## Security Score

| Review | Score |
| --- | --- |
| Overall Score | 87/100 |
| Auditor Score | 82/100 |

| Review by Section | Score |
| --- | --- |
| Manual Scan Score | 45/57 |
| Advance Check Score | 15/19 |

## 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 **Banger**, aiming to uncover potential issues and vulnerabilities within the **Banger** 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** | Banger |
| **Blockchain** | Ethereum |
| **Language** | Solidity |
| **Codebase** | https://etherscan.io/address/0x52a163f4cef6a42fa6b17e0f2a773fc149547c9c |
| **Commit** | 9c7139cb650b00a76c9c5392bac4c1d89ece40778a3ccfa8f1ceb51318a4e942 |

## Audit Summary

| | |
|---|---|
| **Delivery Date** | Nov 11th, 2023 |
| **Audit Methodology** | Static Analysis, Manual Review |
| **Key Components** | Banger.sol |

## Vulnerability Summary

18 Total Findings

| 0 | 0 | 0 | 18 | 0 |
|---|---|---|---|---|
| Resolved | Mitigated | Partially Resolved | Aknowledged | Declined |

| Vulnerability Level | Total | Pending | Declined | Aknowledged | Resolved |
|---|---|---|---|---|---|
| High | 2 | 0 | 0 | 2 | 0 |
| Medium | 1 | 0 | 0 | 1 | 0 |
| Low | 1 | 0 | 0 | 1 | 0 |
| Informational | 14 | 0 | 0 | 14 | 0 |
| Discussion | 0 | 0 | 0 | 0 | 0 |

## Audit Scope

| ID | File | KECCAK256 or SHA256 Checksum |
|---|---|---|
| BNG | Banger.sol | 0x788b3fb99ff6a8040fc5319a9bef95a63aeb2a6cd0b87cbf0850a257d456610c |

# Understandings

$BANGER isn't just a token; it's a gateway to seamlessly bring your company or project brand to life. Leveraging the transformative power of AI, users can generate captivating logos, vibrant Telegram stickers, and compelling marketing materials in a matter of seconds.

## The AI-Powered Design Revolution

BangerBot, the ingenious companion to $BANGER, utilizes custom algorithms to convert your ideas into professional-grade visuals with just a few keystrokes. From crafting distinctive logos to ready-to-use media, this innovative tool simplifies the design process, eliminating the need for specialized graphic design skills.

## Token Information

- Token Name: BANGER
- Symbol: BANGER
- Decimals: 18
- Total Supply: 5,000,000,000 BANGER

## Tax Distribution

Banger transactions incur a total fee, divided into various components:

- Liquidity Fee: Fee for providing liquidity to the token, adjustable by the owner.
- Team Fee: Portion for the project's team, owner-adjustable.
- Marketing Fee: Allocated for marketing efforts, configurable by the owner.
- Dev Fee: Allocated for development, owner-adjustable.
- Banger Fee: Fee for the BANGER ecosystem, owner-adjustable.
- Total Fee: The sum of the above fees used for overall fee distribution.
- Fee Denominator: Denominator used in fee calculations, typically set to 100.

## Fee Management

The owner can manage various fees:

- Set Tax: Configure liquidity, team, marketing, dev, and Banger fees, along with the fee denominator.
- Set Fee Multipliers: Adjust percentage fees for buy, sell, and transfer transactions.
- Set Fee Receivers: Specify addresses receiving various fee components.

## Tax Exemption

The owner can exempt specific addresses from fees, allowing fee exemption for certain addresses, useful for whitelisting wallets or contracts.

## Ownership and Authorization

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

## Transaction Limits

Enforced transaction limits prevent excessive token movement, ensuring users stay within defined limits.

## Swap Mechanism

Banger employs a swap mechanism to manage liquidity. When a token threshold is reached, a portion is swapped to BB tokens via the PancakeSwap Router, affecting the token's price temporarily. The remaining balance is supplied to the BANGER-BNB liquidity pool.

## Open Trading

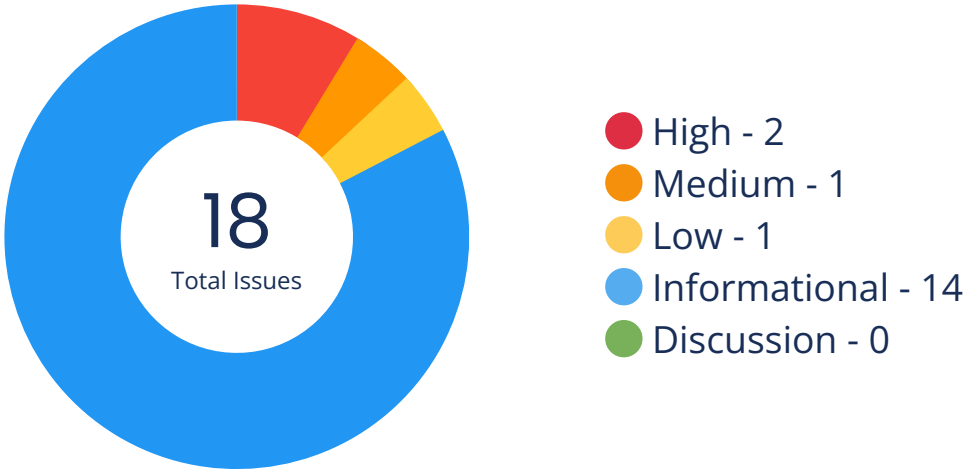
Trading can be restricted based on conditions set by the owner, ensuring it remains closed until specific requirements are met.

## Additional Functionality

The contract includes functions like clearing stuck ETH, clearing tokens, and more.

This understanding provides insights into the key features and functions of the Banger contract, governing various aspects of its operation, including fees, liquidity, and user interactions.

# Findings




| Location | Title | Scope | Severity | Status |
|---|---|---|---|---|
| Banger.sol:1034,1036 | Freeze Money | Banger | High | Aknowledged |
| Banger.sol:1036 | Reentrancy | Banger | High | Aknowledged |
| Banger.sol:1042 | Unchecked Call Return Value | Banger | Medium | Aknowledged |
| Banger.sol:1042 | Use Safer Functions | Banger | Low | Aknowledged |
| Banger.sol:656 | Set the Constant to Private | Banger | Informational | Aknowledged |
| Banger.sol:1,16,368,567,626 | Recommend to Follow Code Layout Conventions | IERC20 | Informational | Aknowledged |
| Banger.sol:973,1036 | No Check of Address Params with Zero Address | Banger | Informational | Aknowledged |
| Banger.sol:968 | Use ++i/--i Instead of i++/i-- | Banger | Informational | Aknowledged |
| Banger.sol:320,323,345 | Continuous State Variable Write | ERC20 | Informational | Aknowledged |

| Location | Title | Scope | Severity | Status |
|---|---|---|---|---|
| Banger.sol:750,1041,1042 | Cache State Variables that are Read Multiple Times within A Function | Banger | Informational | Aknowledged |
| Banger.sol:132,140,183,206,228,247,267,379,393 | Function Visibility Can Be External | ERC20 | Informational | Aknowledged |
| Banger.sol:270,293,294,297,318,342,343,360,384,394,581,705,890,891,897,899,906,912,918,919,929,940,946,951,967,974,975,981,986,991,1037 | Use CustomError Instead of String | ERC20 | Informational | Aknowledged |
| Banger.sol:1037 | Lack of Error Message | Banger | Informational | Aknowledged |
| Banger.sol:270,293,294,297,342,343,394,581,918,919,975 | Long String in revert/require | ERC20 | Informational | Aknowledged |
| Banger.sol:646,647 | Variables Can Be Declared as Immutable | Banger | Informational | Aknowledged |
| Banger.sol:1037 | Get Contract Balance of ETH in Assembly | Banger | Informational | Aknowledged |
| Banger.sol:293,294,318,342,343,394,890,897,974,981,986,991 | Use Assembly to Check Zero Address | ERC20 | Informational | Aknowledged |
| Banger.sol:722,918 | Too Many Digits | Banger | Informational | Aknowledged |

## Code Security - Freeze Money

| Title | Severity | Location | Status |
| --- | --- | --- | --- |
| Freeze Money | High | Banger.sol:1034,1036 | 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 - Reentrancy

| Title | Severity | Location | Status |
| --- | --- | --- | --- |
| Reentrancy | High | Banger.sol:1036 | Aknowledged |

### Description

As the external call is executed prior to state variables alterations, it exposes the possibility for the external contract to perform a reentrancy attack by calling back into this contract.

## Code Security - Unchecked Call Return Value

| Title | Severity | Location | Status |
| --- | --- | --- | --- |
| Unchecked Call Return Value | Medium | Banger.sol:1042 | Aknowledged |

### Description

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

## Code Security - Use Safer Functions

| Title | Severity | Location | Status |
| --- | --- | --- | --- |
| Use Safer Functions | Low | Banger.sol:1042 | 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 - Set the Constant to Private

| Title | Severity | Location | Status |
| --- | --- | --- | --- |
| Set the Constant to Private | Informational | Banger.sol:656 | 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 | Banger.sol:1,16,368,567,626 | 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 - No Check of Address Params with Zero Address

| Title | Severity | Location | Status |
| --- | --- | --- | --- |
| No Check of Address Params with Zero Address | Informational | Banger.sol:973,1036 | Aknowledged |

### Description

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

## Optimization Suggestion - Use ++i/--i Instead of i++/i--

| Title | Severity | Location | Status |
| --- | --- | --- | --- |
| Use ++i/--i Instead of i++/i-- | Informational | Banger.sol:968 | 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 - Continuous State Variable Write

| Title | Severity | Location | Status |
| --- | --- | --- | --- |
| Continuous State Variable Write | Informational | Banger.sol:320,323,345 | 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 | Informational | Banger.sol:750,1041,1042 | 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 | Banger.sol:132,140,183,206,228,247,267,379,393 | 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 | Banger.sol:270,293,294,297,318,342,343,360,384,394,581,705,890,891,897,899,906,912,918,919,929,940,946,951,967,974,975,981,986,991,1037 | Aknowledged |

### Description

When using require or revert, CustomError is more gas efficient than string description, as the error message described using CustomError is only compiled into four bytes. Especially when string exceeds 32 bytes, more gas will be consumed. Generally, around 250-270 gas can be saved for one CustomError replacement when compiler optimization is turned off, 60-80 gas can be saved even if compiler optimization is turned on.

## Optimization Suggestion - Lack of Error Message

| Title | Severity | Location | Status |
| --- | --- | --- | --- |
| Lack of Error Message | Informational | Banger.sol:1037 | Aknowledged |

### Description

Use empty string as parameter while invoking function Revert() or Require().

## Optimization Suggestion - Long String in revert/require

| Title | Severity | Location | Status |
| --- | --- | --- | --- |
| Long String in revert/require | Informational | Banger.sol:270,293,294,297,342,343,394,581,918,919,975 | 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 | Banger.sol:646,647 | 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 | Banger.sol:1037 | 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 | Banger.sol:293,294,318,342,343,394,890,897,974,981,986,991 | Aknowledged |

### Description

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

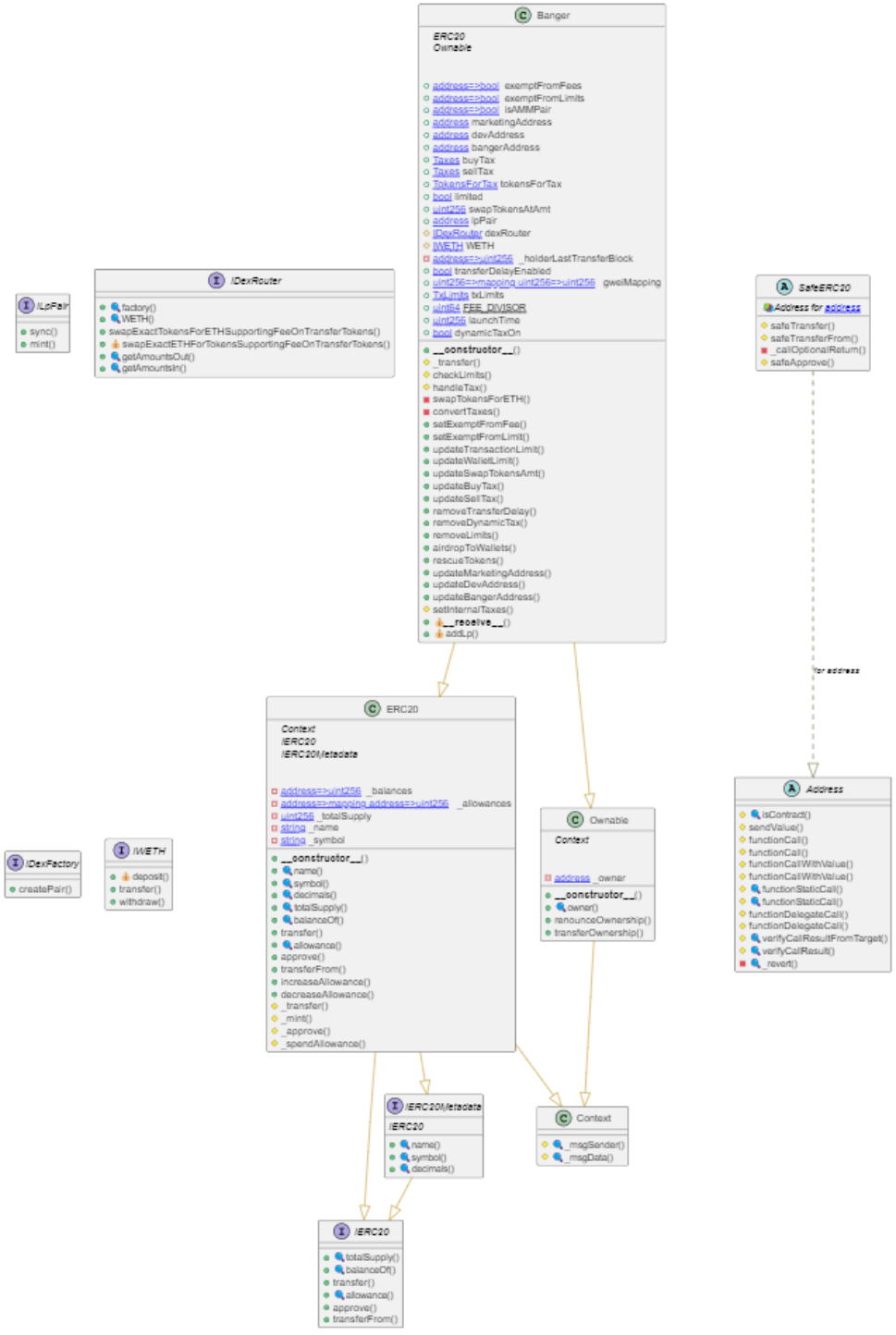
## Optimization Suggestion - Too Many Digits

| Title | Severity | Location | Status |
| --- | --- | --- | --- |
| Too Many Digits | Informational | Banger.sol:722,918 | Aknowledged |

### Description

The number is too long, and it is easy to make mistakes when modifying and maintaining.

# PlantUML


Banger
ERC20
Ownable
address=>bool exemptFromFees
address=>bool exemptFromLimits
address=>bool isAMMPair
address marketingAddress
address devAddress
address bangerAddress
Taxes buyTax
Taxes sellTax
TokensForTax tokensForTax
bool limited
uint256 swapTokensAtAmt
address lpPair
IDexRouter dexRouter
IWETH WETH
address=>uint256 _holderLastTransferBlock
bool transferDelayEnabled
uint256=>mapping uint256=>uint256 gweiMapping
TxLimits txLimits
uint64 FEE_DIVISOR
uint256 launchTime
bool dynamicTaxOn
__constructor__()
_transfer()
checkLimits()
handleTax()
swapTokensForETH()
convertTaxes()
setExemptFromFee()
setExemptFromLimit()
updateTransactionLimit()
updateWalletLimit()
updateSwapTokensAmt()
updateBuyTax()
updateSellTax()
removeTransferDelay()
removeDynamicTax()
removeLimits()
airdropToWallets()
rescueTokens()
updateMarketingAddress()
updateDevAddress()
updateBangerAddress()
setInternalTaxes()
__receive__()
addLp()
IDexRouter
factory()
WETH()
swapExactTokensForETHSupportingFeeOnTransferTokens()
swapExactETHForTokensSupportingFeeOnTransferTokens()
getAmountsOut()
getAmountsIn()
ILpPair
sync()
mint()
SafeERC20
Address for address
safeTransfer()
safeTransferFrom()
_callOptionalReturn()
safeApprove()
for address
Address
isContract()
sendValue()
functionCall()
functionCall()
functionCallWithValue()
functionCallWithValue()
functionStaticCall()
functionStaticCall()
functionDelegateCall()
functionDelegateCall()
verifyCallResultFromTarget()
verifyCallResult()
_revert()
ERC20
Context
IERC20
IERC20Metadata
address=>uint256 _balances
address=>mapping address=>uint256 _allowances
uint256 _totalSupply
string _name
string _symbol
__constructor__()
name()
symbol()
decimals()
totalSupply()
balanceOf()
transfer()
allowance()
approve()
transferFrom()
increaseAllowance()
decreaseAllowance()
_transfer()
_mint()
_approve()
_spendAllowance()
Ownable
Context
address _owner
__constructor__()
owner()
renounceOwnership()
transferOwnership()
IDexFactory
createPair()
IWETH
deposit()
transfer()
withdraw()
IERC20Metadata
IERC20
name()
symbol()
decimals()
Context
_msgSender()
_msgData()
IERC20
totalSupply()
balanceOf()
transfer()
allowance()
approve()
transferFrom()

# Appendix

## Finding Categories

### Security and Best Practices

1. Freeze Money: Freezing funds without appropriate conditions or control mechanisms poses a serious security risk. Consider implementing time locks or multi-signature requirements to prevent unauthorized freezing of assets.
2. Reentrancy: Guard against reentrancy attacks by using the 'reentrancyGuard' modifier or using the 'nonReentrant' modifier from the ReentrancyGuard library.
3. Unchecked Call Return Value: Always check the return value of external calls to prevent unexpected behavior. Utilize checks-effects-interactions pattern to ensure secure interactions with external contracts.
4. Use Safer Functions: Prioritize the use of secure and well-audited functions to mitigate potential security vulnerabilities. Review functions thoroughly to enhance overall security.
5. Set the Constant to Private: Declaring constants as private ensures that they cannot be accessed externally, preventing unwanted external access and potential security risks.
6. Recommend to Follow Code Layout Conventions: Strict adherence to established code layout conventions significantly improves code readability and maintainability. Adopt a consistent coding style throughout the contract.
7. No Check of Address Params with Zero Address: Verify address parameters to ensure they are not set to the zero address. Failing to check this condition can lead to unexpected behavior and potential vulnerabilities.
8. Use ++i/--i Instead of i++/i--: Prefer using pre-increment (++) and pre-decrement (--) operators for better gas efficiency and to avoid unnecessary state changes.
9. Continuous State Variable Write: Minimize state variable writes within a function to optimize gas usage. Limiting unnecessary state changes enhances contract efficiency.
10. 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 improves overall contract efficiency.
11. 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 called more cost-effectively.
12. Use CustomError Instead of String: Opt for custom error codes rather than string error messages for more efficient contract operation. Custom errors reduce gas costs and improve contract efficiency.
13. Lack of Error Message: Include informative error messages in revert or require statements to provide clarity on the cause of failures. Clear error messages assist in debugging and enhance user experience.
14. Long String in revert/require: Optimize gas usage by minimizing the length of revert or require strings. Long strings increase gas costs and should be optimized for efficiency.
15. Variables Can Be Declared as Immutable: Declare variables as immutable when they don't change after initialization. Immutable variables enhance security and readability by preventing unintentional modifications.
16. Get Contract Balance of ETH in Assembly: Optimize gas usage by using assembly to get the contract balance of ETH. Assembly provides a more efficient way to perform certain operations.
17. Use Assembly to Check Zero Address: Leverage optimized assembly checks to verify zero addresses efficiently. Assembly can provide a gas-efficient solution for certain address-related validations.
18. Too Many Digits: Avoid using excessive decimal places, as it can lead to precision errors and unexpected behavior. Stick to a reasonable number of decimal places for better contract stability and usability.

# 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://itsabanger.io/

**Network Security**

**High** | 0 Attentions

**Application Security**

**High** | 3 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

8 Passed | 3 Attention

| | |
|---|---|
| Missing X-Frame-Options Header | YES |
| Missing HSTS header | NO |
| Missing X-Content-Type-Options Header | YES |
| Missing Content Security Policy (CSP) | YES |
| HTTP Access Allowed | NO |
| Self-Signed Certificate | NO |
| Wrong Host Certificate | NO |
| Expired Certificate | NO |
| SSL/TLS Supports Weak Cipher | NO |
| Support SSL Protocols | NO |
| Support TLS Weak Version | NO |

## DNS Health

7 Passed | 3 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 | 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 | 2023.11.08 20:00 (UTC) |
| Token Market Cap (estimate) | $410.92K |
| Token/Project Age | 2 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.