



CFG NINJA AUDITS

Security Assessment

ZKChain Staking

May 15, 2023

Audit Status: Pass

Audit Edition: Advance



POWERED BY
BLADE POOL

Table of Contents

1 Assessment Summary

2 Technical Findings Summary

3 Project Overview

3.1 Main Contract Assessed

4 Smart Contract Risk Checks

5 Contract Ownership

7 KYC Check

8 Smart Contract Vulnerability Checks

8.1 Smart Contract Vulnerability Details

8.2 Smart Contract Inheritance Details

8.3 Smart Contract Privileged Functions

9 Assessment Results and Notes(Important)

10 Social Media Check(Informational)

11 Technical Findings Details

12 Disclaimer



Assessment Summary

This report has been prepared for ZKChain Staking on the Binance Smart Chain network. CFGNINJA provides both client-centered and user-centered examination of the smart contracts and their current status when applicable. This report represents the security assessment made to find issues and vulnerabilities on the source code along with the current liquidity and token holder statistics of the protocol.

A comprehensive examination has been performed, utilizing Cross Referencing, Static Analysis, In-House Security Tools, and line-by-line Manual Review.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Inspecting liquidity and holders statistics to inform the current status to both users and client when applicable.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Verifying contract functions that allow trusted and/or untrusted actors to mint, lock, pause, and transfer assets.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders
- Thorough line-by-line manual review of the entire codebase by industry experts.



Project Overview

Token Summary

| Parameter | Result |
|---------------|---|
| Address | 0xEaAD0A4C8cfE770541F45BF73AA8AC2e8B4671d4 |
| Name | ZKChain |
| Token Tracker | ZKChain (Staking) |
| Decimals | 18 |
| Supply | 1,000,000,000 |
| Platform | Binance Smart Chain |
| compiler | v0.8.19+commit.7dd6d404 |
| Contract Name | ZKCLPFarm |
| Optimization | Yes with 200 runs |
| LicenseType | MIT |
| Language | Solidity |
| Codebase | https://bscscan.com/address/0xEaAD0A4C8cfE770541F45BF73AA8AC2e8B4671d4#code |
| Payment Tx | 0x7c01448176f2e9c940183803a4b648038cc99df24f0b796575b3ddf549ebe89f |



Main Contract Assessed Contract Name

| Name | Contract | Live |
|---------|--|------|
| ZKChain | 0xEaAD0A4C8cfE770541F45BF73AA8AC2e8B4671d4 | Yes |

TestNet Contract Assessed Contract Name

| Name | Contract | Live |
|---------|--|------|
| ZKChain | 0xD3AEbB3178ff400A25b8dAC2095A0a6dC3119C57 | Yes |

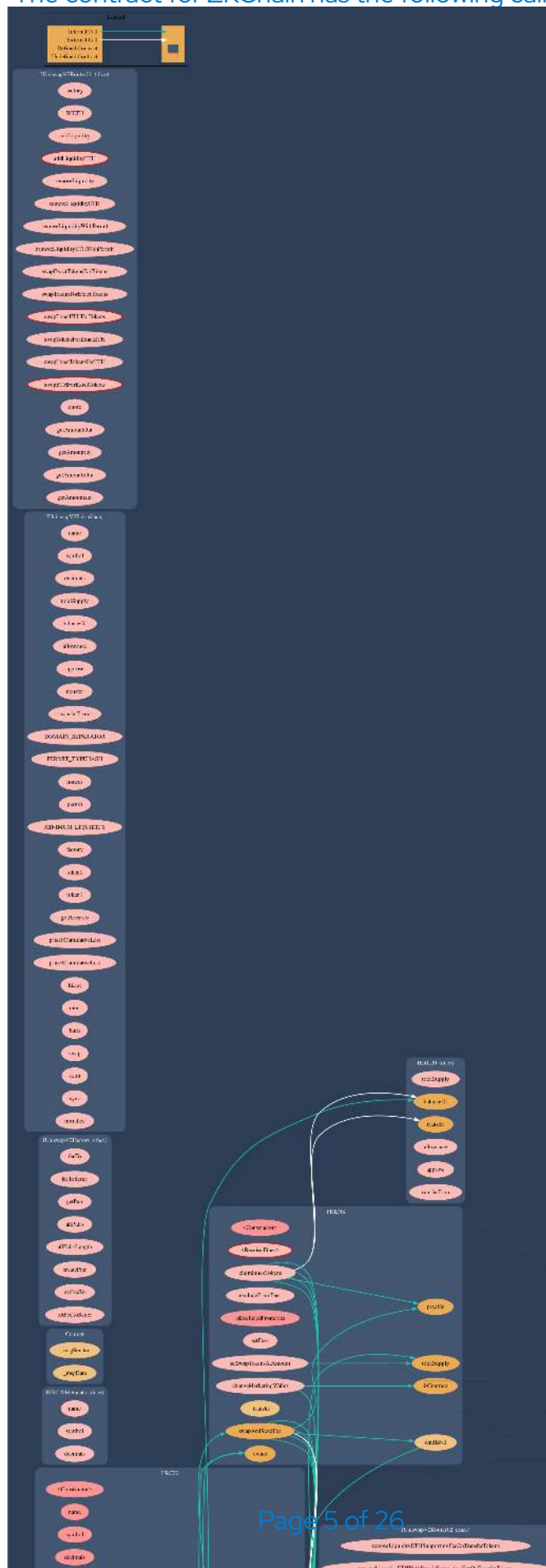
Solidity Code Provided

| SolID | File Sha-1 | FileName |
|-----------|--|---------------|
| ZKCLPFarm | c0ec8a4207eda547ab8faee418e0ebb0d7efe212 | ZKCLPFarm.sol |



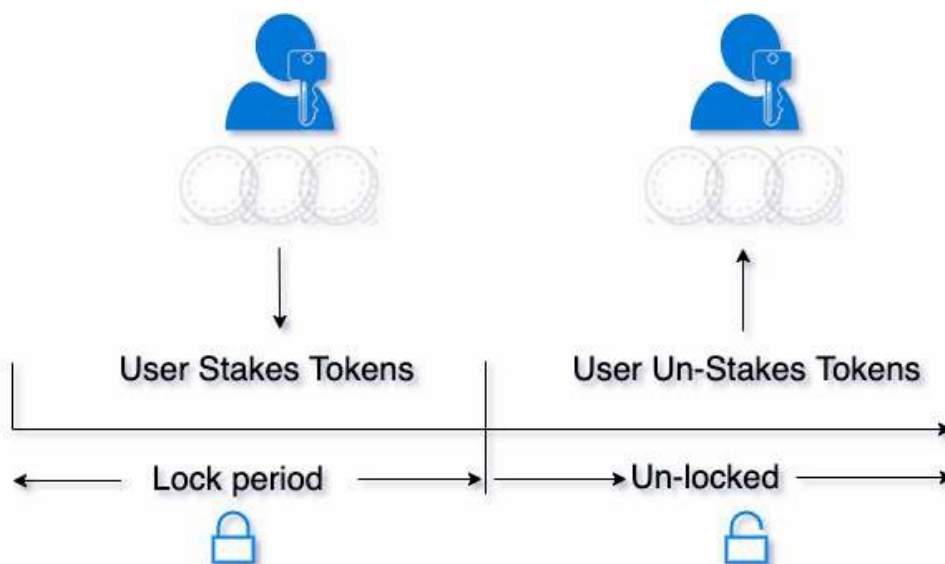
Call Graph

The contract for ZKChain has the following call graph structure.



What is a Staking Contract

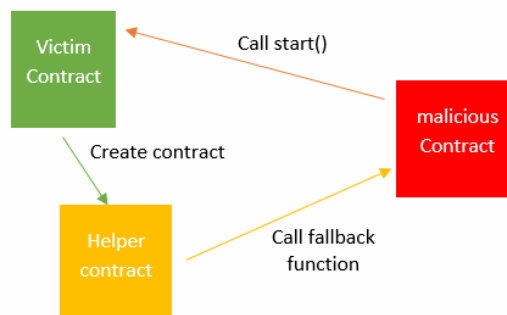
A smart contract which allows users to stake and un-stake a specified ERC20 token. Staked tokens are locked for a specific length of time (set by the contract owner at the outset). Once the time period has elapsed, the user can remove their tokens again.



The Project Owners of ZKChain have implemented Reentrancy Guard Library

The Team has done a great job to avoid potential reentrancy issues in the contract.

You can read more about the reentrancy library used.
[ReentrancyGuard](#)



KYC Information

The Project Owners of ZKChain is not KYC.

KYC Information Notes:

Auditor Notes: KYC to be completed by PinkSale, project will be a SAFU Project.

Project Owner Notes:



Smart Contract Vulnerability Checks

The Smart Contract Weakness Classification Registry (SWC Registry) is an implementation of the weakness classification scheme proposed in EIP-1470. It is loosely aligned to the terminologies and structure used in the Common Weakness Enumeration (CWE) while overlaying a wide range of weakness variants that are specific to smart contracts.

| ID | Severity | Name | File | location |
|---------|----------|---|---------------|-----------|
| SWC-100 | Pass | Function Default Visibility | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-101 | Pass | Integer Overflow and Underflow. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-102 | Pass | Outdated Compiler Version file. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-103 | Low | A floating pragma is set. | ZKCLPFarm.sol | L: 7 C: 0 |
| SWC-104 | Pass | Unchecked Call Return Value. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-105 | Pass | Unprotected Ether Withdrawal. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-106 | Pass | Unprotected SELFDESTRUCT Instruction | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-107 | Pass | Read of persistent state following external call. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-108 | Pass | State variable visibility is not set.. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-109 | Pass | Uninitialized Storage Pointer. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-110 | Pass | Assert Violation. | ZKCLPFarm.sol | L: 0 C: 0 |



| ID | Severity | Name | File | location |
|---------|----------|--|---------------|--------------|
| SWC-111 | Pass | Use of Deprecated Solidity Functions. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-112 | Pass | Delegate Call to Untrusted Callee. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-113 | Pass | Multiple calls are executed in the same transaction. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-114 | Pass | Transaction Order Dependence. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-115 | Pass | Authorization through tx.origin. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-116 | Pass | A control flow decision is made based on The block.timestamp environment variable. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-117 | Pass | Signature Malleability. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-118 | Pass | Incorrect Constructor Name. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-119 | Pass | Shadowing State Variables. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-120 | Low | Potential use of block.number as source of randomness. | ZKCLPFarm.sol | L: 818 C: 21 |
| SWC-121 | Pass | Missing Protection against Signature Replay Attacks. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-122 | Pass | Lack of Proper Signature Verification. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-123 | Pass | Requirement Violation. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-124 | Pass | Write to Arbitrary Storage Location. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-125 | Pass | Incorrect Inheritance Order. | ZKCLPFarm.sol | L: 0 C: 0 |



| ID | Severity | Name | File | location |
|---------|----------|--|---------------|-----------|
| SWC-126 | Pass | Insufficient Gas Griefing. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-127 | Pass | Arbitrary Jump with Function Type Variable. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-128 | Pass | DoS With Block Gas Limit. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-129 | Pass | Typographical Error. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-130 | Pass | Right-To-Left-Override control character (U+202E). | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-131 | Pass | Presence of unused variables. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-132 | Pass | Unexpected Ether balance. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-133 | Pass | Hash Collisions with Multiple Variable Length Arguments. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-134 | Pass | Message call with hardcoded gas amount. | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-135 | Pass | Code With No Effects (Irrelevant/Dead Code). | ZKCLPFarm.sol | L: 0 C: 0 |
| SWC-136 | Pass | Unencrypted Private Data On-Chain. | ZKCLPFarm.sol | L: 0 C: 0 |

We scan the contract for additional security issues using MYTHX and industry-standard security scanning tools.



Smart Contract Vulnerability Details

SWC-103 - Floating Pragma.

CWE-664: Improper Control of a Resource Through its Lifetime.

References:

Description:

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation:

Lock the pragma version and also consider known bugs (<https://github.com/ethereum/solidity/releases>) for the compiler version that is chosen.

Pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package. Otherwise, the developer would need to manually update the pragma in order to compile locally.

References:

Ethereum Smart Contract Best Practices - Lock pragmas to specific compiler version.



Smart Contract Vulnerability Details

SWC-120 – Weak Sources of Randomness from Chain Attributes

CWE-330: Use of Insufficiently Random Values

Description:

Solidity allows for ambiguous naming of state variables when inheritance is used. Contract A with a variable x could inherit contract B that also has a state variable x defined. This would result in two separate versions of x, one of them being accessed from contract A and the other one from contract B. In more complex contract systems this condition could go unnoticed and subsequently lead to security issues.

Shadowing state variables can also occur within a single contract when there are multiple definitions on the contract and function level.

Remediation:

Using commitment scheme, e.g. RANDAO. Using external sources of randomness via oracles, e.g. Oraclize. Note that this approach requires trusting in oracle, thus it may be reasonable to use multiple oracles. Using Bitcoin block hashes, as they are more expensive to mine.

References:

How can I securely generate a random number in my smart contract?)

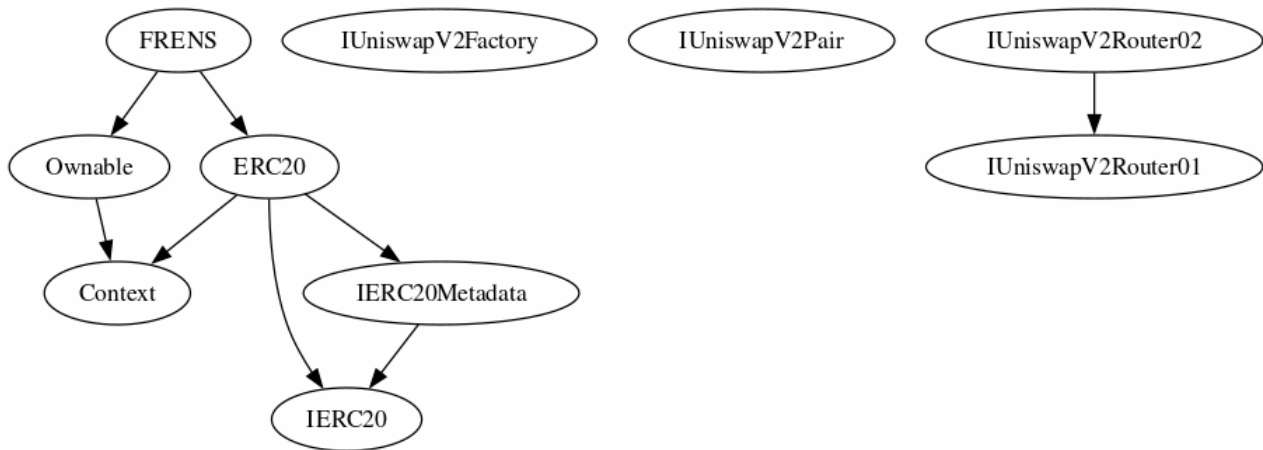
When can BLOCKHASH be safely used for a random number? When would it be unsafe?

The Run smart contract.



Inheritance

The contract for ZKChain has the following inheritance structure.



Privileged Functions (onlyOwner)

Please Note if the contract is Renounced none of this functions can be executed.

| Function Name | Parameters | Visibility |
|---------------------|---|------------|
| renounceOwnership | | Public |
| transferOwnership | address newOwner | Public |
| recoverTreasure | IBEP20 recoverToken, uint256 amount | External |
| addRewardTreasure | uint256 _amount, uint256 _tokenPerBlock | External |
| setEarlyWithdrawFee | uint256 _earlyFeePercent | External |
| setMarketingAddress | address _marketingAddress | External |



Smart Contract Advance Checks



| ID | Severity | Name | Result | Status |
|------------|---------------|--|--------|-----------|
| Staking-01 | Minor | Potential Sandwich Attacks. | Pass | Not-Found |
| Staking-02 | Minor | Function Visibility Optimization | Pass | Not-found |
| Staking-03 | Minor | Lack of Input Validation. | Fail | Pending |
| Staking-04 | Major | Centralized Risk In addLiquidity. | Pass | Not-found |
| Staking-05 | Minor | Missing Event Emission. | Fail | Pending |
| Staking-06 | Minor | Conformance with Solidity Naming Conventions. | Pass | Not-found |
| Staking-07 | Minor | State Variables could be Declared Constant. | Pass | Not-Found |
| Staking-08 | Minor | Dead Code Elimination. | Pass | Not-Found |
| Staking-09 | Major | Third Party Dependencies. | Pass | Not-Found |
| Staking-10 | Major | Initial Token Distribution. | Pass | Not-found |
| Staking-11 | Major | multiTransfer is present within the contract. | Pass | Not-Found |
| Staking-12 | Major | Centralization Risks In The X Role | Pass | Resolved |
| Staking-13 | Informational | Extra Gas Cost For User.. | Pass | Not-found |
| Staking-14 | Medium | Unnecessary Use Of SafeMath | Fail | Pending |
| Staking-15 | Medium | Symbol Length Limitation due to Solidity Naming Standards. | Pass | Not-Found |



| ID | Severity | Name | Result | Status |
|------------|---------------|---|--------|-----------|
| Staking-16 | Medium | Invalid collection of Taxes during Transfer. | Pass | Not-Found |
| Staking-17 | Informational | Conformance to numeric notation best practice. | Pass | Not-Found |
| Staking-18 | Informational | Enable Trade and Exclude Exist to create a whitelist. | Pass | Not-Found |



Staking-03 | Lack of Input Validation.

| Category | Severity | Location | Status |
|---------------|---|-----------------------|---|
| Volatile Code |  Minor | ZKCLPFarm.sol: 849,14 |  Pending |

Description

The given input is missing the check for the non-zero address.

The given input is missing the check for the setMarketingAddress, deposit, withdraw, emergencyWithdraw is missing required function.

Remediation



We advise the client to add the check for the passed-in values to prevent unexpected errors as below:

```
...
require(receiver != address(0), "Receiver is the zero address");
...
...
require(value X limitation, "Your not able to do this function");
...
```

We also recommend customer to review the following function that is missing a required validation. setMarketingAddress, deposit, withdraw, emergencyWithdraw is missing required function.



Staking-05 | Missing Event Emission.

| Category | Severity | Location | Status |
|---------------|---|-------------------------|---|
| Volatile Code |  Minor | ZKCLPFarm.sol: 1077, 14 |  Pending |

Description



Detected missing events for critical arithmetic parameters. There are functions that have no event emitted, so it is difficult to track off-chain changes. The linked code does not create an event for the transfer.

Remediation

Emit an event for critical parameter changes. It is recommended emitting events for the sensitive functions that are controlled by centralization roles.



Staking-14 | Unnecessary Use Of SafeMath

| Category | Severity | Location | Status |
|---------------|--|---------------------|---|
| Logical Issue |  Medium | ZKCLPFarm.sol: 22,9 |  Pending |

Description

The SafeMath library is used unnecessarily. With Solidity compiler versions 0.8.0 or newer, arithmetic operations

will automatically revert in case of integer overflow or underflow.

library SafeMath {

An implementation of SafeMath library is found.

using SafeMath for uint256;

SafeMath library is used for uint256 type in contract.

_balances[recipient] = _balances[recipient].add(amount);

magnifiedDividendPerShare = magnifiedDividendPerShare.add(
(amount).mul(magnitude) / totalSupply()

);

Note: Only a sample of 2 SafeMath library usage in this contract (out of 14) are shown above.

Remediation

We advise removing the usage of SafeMath library and using the built-in arithmetic operations provided by the

Solidity programming language

Project Action








Technical Findings Summary

Classification of Risk

| Severity | Description |
|---|--|
|  Critical | Risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks. |
|  Major | Risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project. |
|  Medium | Risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform |
|  Minor | Risks can be any of the above but on a smaller scale. They generally do not compromise the overall integrity of the Project, but they may be less efficient than other solutions. |
|  Informational | Errors are often recommended to improve the code's style or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code. |

Findings

| Severity | Found | Pending | Resolved |
|---|-------|---------|----------|
|  Critical | 0 | 0 | 0 |
|  Major | 1 | 0 | 0 |
|  Medium | 0 | 0 | 0 |
|  Minor | 2 | 0 | 0 |
|  Informational | 0 | 0 | 0 |
| Total | 3 | 0 | 0 |



Social Media Checks

| Social Media | URL | Result |
|--------------|---|--------|
| Twitter | https://twitter.com/zk_chain | Pass |
| Other | https://zk-chain.medium.com/ | Pass |
| Website | https://www.zk-chain.org/ | Pass |
| Telegram | https://t.me/ZK_Chain_Group | Pass |

We recommend to have 3 or more social media sources including a completed working websites.

Social Media Information Notes:

Auditor Notes: undefined

Project Owner Notes:



Audit Result

Final Audit Score

| Review | Score |
|----------------|-------|
| Security Score | 85 |
| Auditor Score | 80 |

The Following Score System Has been Added to this page to help understand the value of the audit, the maximum score is 100, however to attain that value the project must pass and provide all the data needed for the assessment. Our Passing Score has been changed to 80 Points, if a project does not attain 80% is an automatic failure. Read our notes and final assessment below.

Audit Passed



Assessment Results

Important Notes:

- No issues or vulnerabilities were found.
- The contract was tested and is fully functional.
- the following contract has been used for a few staking platforms in the past and is stable, however there are a few coding improvements that can be done.

Auditor Score =80
Audit Passed



Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different requirements on the input variables than a setter function.

Coding Best Practices

ERC 20 Coding Standards are a set of rules that each developer should follow to ensure the code meets a set of criteria and is readable by all the developers.



Disclaimer

CFGNINJA has conducted an independent security assessment to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the reviewed code for the scope of this assessment. This report does not constitute agreement, acceptance, or advocacy for the Project, and users relying on this report should not consider this as having any merit for financial advice in any shape, form, or nature. The contracts audited do not account for any economic developments that the Project in question may pursue, and the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude, and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are entirely free of exploits, bugs, vulnerabilities or deprecation of technologies.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence, regardless of the findings presented. Information is provided 'as is, and CFGNINJA is under no covenant to audited completeness, accuracy, or solidity of the contracts. In no event will CFGNINJA or its partners, employees, agents, or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions or actions with regards to the information provided in this audit report.

The assessment services provided by CFGNINJA are subject to dependencies and are under continuing development. You agree that your access or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies with high levels of technical risk and uncertainty. The assessment reports could include false positives, negatives, and unpredictable results. The services may access, and depend upon, multiple layers of third parties.

