CHAINHAPPY (CIH)

SMART CONTRACT AUDIT REPORT

Prepared by:

BlockAudit

Date Of Enrollment:

March 08th, 2022 - March 17th, 2022

Visit : www.blockaudit.report

TABLE OF CONTENTS

SUMMARY

This Audit Report mainly focuses on the extensive security of **CHAINHAPPY (CIH)** Smart Contracts. With this report, we attempt to ensure the reliability and correctness of the smart contract by complete and rigorous assessment of the system's architecture and the smart contract codebase.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts producedby industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified inpublic;
- Provide more transparency on privileged activities once the protocol is live.

OVERVIEW

Project Summary

Project Name	CHAINHAPPY (CIH)
Logo	
Platform	BSC
Language	Solidity
Code Link	https://bscscan.com/address/0x1Dfc4E1057E088b66B7e09EE9cE72Ebc17EFf788#code

File Summary

ID	File Name	Audit Status
CIH	CIHToken.sol	Failed

Audit Summary

Date of Delivery	17 Mar 2022
Audit Methodology	Code Analysis. Automatic Assesment, Manual Review
Audit Result	Failed ✕
Audit Team	BlockAudit Report Team

FINDINGS

Critical	0	0.0%
High	0	0.0%
Medium	1	7.69%
Low	4	30.76%
Informational	0	0.0%
Ownership	0	0.0%
Gas Optimization	8	61.53%

Vulnerability Findings Summary

ID	Type	Instances	Severity	Status
CIH01	Centralization Risk For Trusted Owners	4	Medium	Reported
CIH02	Unsafe ERC20 operation(s)	3	Low	Reported
CIH03	Missing Checks For Address(0) When Assigning Values To Address State Variables	1	Low	Reported
CIH04	Return Values Of Approve() Not Checked	4	Low	Reported
CIH05	Event Is Missing Indexed Fields	2	Low	Reported
CIH06	Use Assembly To Check For Address(0)	7	Gas Optimization	Reported
CIH07	State Variables Should Be Cached In Stack Variables Rather Than Re-Reading Them From Storage	1	Gas Optimization	Reported
CIH08	Use calldata instead of memory for function arguments that do not get mutated	2	Gas Optimization	Reported
CIH09	Use Custom Errors	12	Gas Optimization	Reported
CIH10	Long Revert Strings	10	Gas Optimization	Reported
CIH11	Functions guaranteed to revert when called by normal users can be marked payable	2	Gas Optimization	Reported
CIH12	Use != 0 instead of > 0 for unsigned integer comparison	2	Gas Optimization	Reported
CIH13	INTERNAL FUNCTIONS NOT CALLED BY THE CONTRACT SHOULD BE REMOVED	13	Gas Optimization	Reported

CIH01

Type	Centralization Risk for trusted owners
Severity	Medium
File	CIHTOKEN.sol
Instances	4
Status	**Reported**

Description

Contracts have owners with privileged rights to perform admin tasks and need to be trusted to not perform malicious updates or drain funds.

Snapshot

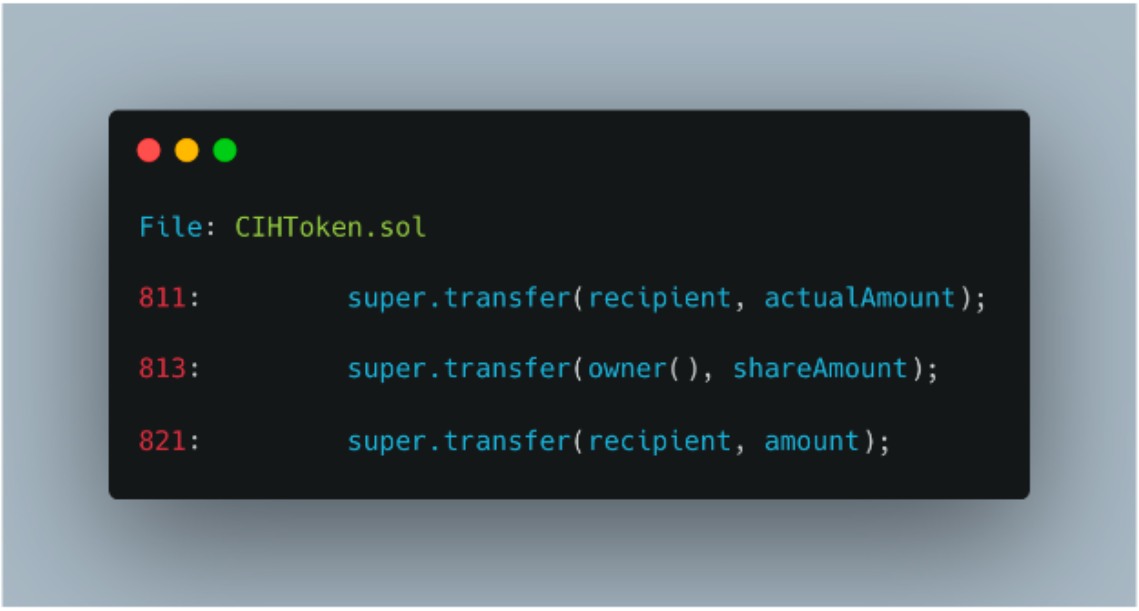
CIH02

Type	Unsafe ERC20 operation(s)
Severity	Low
File	CIHTOKEN.sol
Instances	3
Status	**Reported**

Description

Unsafe ERC20 operation(s)

Snapshot



CIH03

Type	Missing checks for address(0) when assigning values to address state variables
Severity	Low
File	CIHTOKEN.sol
Instances	10
Status	**Reported**

Description

Missing checks for address(0) when assigning values to address state variables

Snapshot

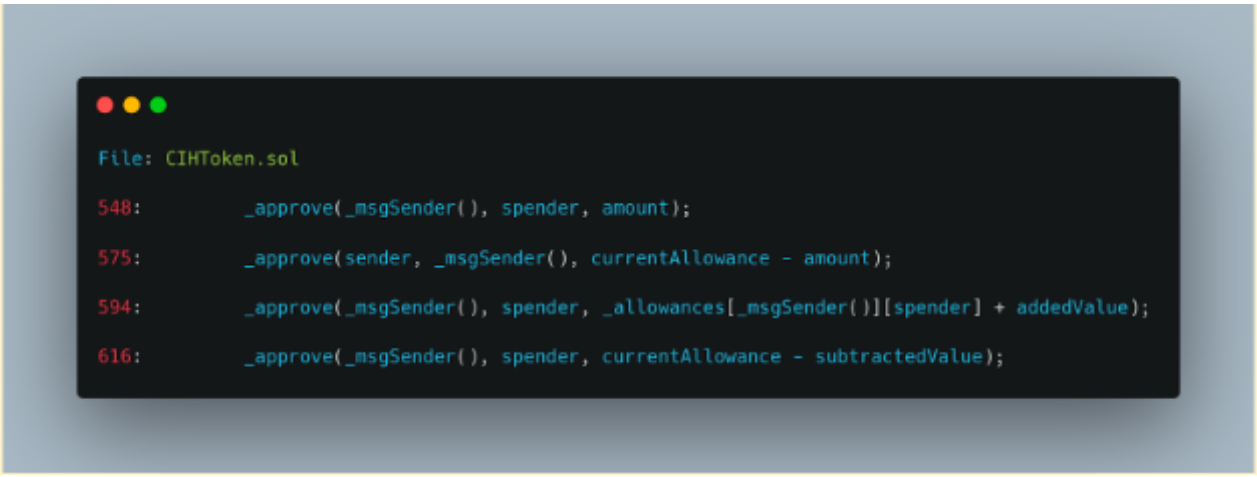
CIH04

Type	Return values of approve() not checked
Severity	Low
File	CIHTOKEN.sol
Instances	4
Status	**Reported**

Description

Not all IERC20 implementations revert() when there's a failure in approve(). The function signature has a boolean return value and they indicate errors that way instead. By not checking the return value, operations that should have marked as failed, may potentially go through without actually approving anything

Snapshot



```
File: CIHToken.sol
548:        _approve(_msgSender(), spender, amount);
575:        _approve(sender, _msgSender(), currentAllowance - amount);
594:        _approve(_msgSender(), spender, _allowances[_msgSender()][spender] + addedValue);
616:        _approve(_msgSender(), spender, currentAllowance - subtractedValue);
```

CIH05

Type	Event is missing indexed fields
Severity	Low
File	CIHTOKEN.sol
Instances	2
Status	**Reported**

Description

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Each event should use three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

Snapshot

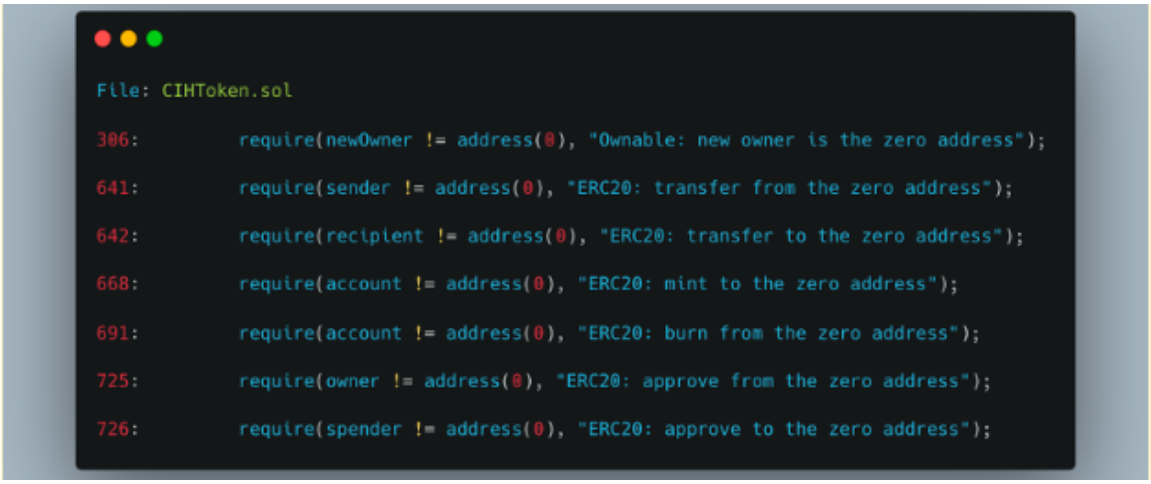
CIH06

Type	Use assembly to check for address(0)
Severity	Gas Optimisation
File	CIHTOKEN.sol
Instances	7
Status	**Reported**

Description

Saves 6 gas per instance

Snapshot



```
File: CIHToken.sol

306:        require(newOwner != address(0), "Ownable: new owner is the zero address");

641:        require(sender != address(0), "ERC20: transfer from the zero address");

642:        require(recipient != address(0), "ERC20: transfer to the zero address");

668:        require(account != address(0), "ERC20: mint to the zero address");

691:        require(account != address(0), "ERC20: burn from the zero address");

725:        require(owner != address(0), "ERC20: approve from the zero address");

726:        require(spender != address(0), "ERC20: approve to the zero address");
```

CIH07

Type	State variables should be cached in stack variables rather than re-reading them from storage
Severity	Gas Optimisation
File	CIHTOKEN.sol
Instances	1
Status	**Reported**

Description

The instances below point to the second+ access of a state variable within a function. Caching of a state variable replaces each Gwarmaccess (100 gas) with a much cheaper stack read. Other less obvious fixes/optimizations include having local memory caches of state variable structs, or having local caches of state variable contracts/addresses.

Saves 100 gas per instance

Snapshot

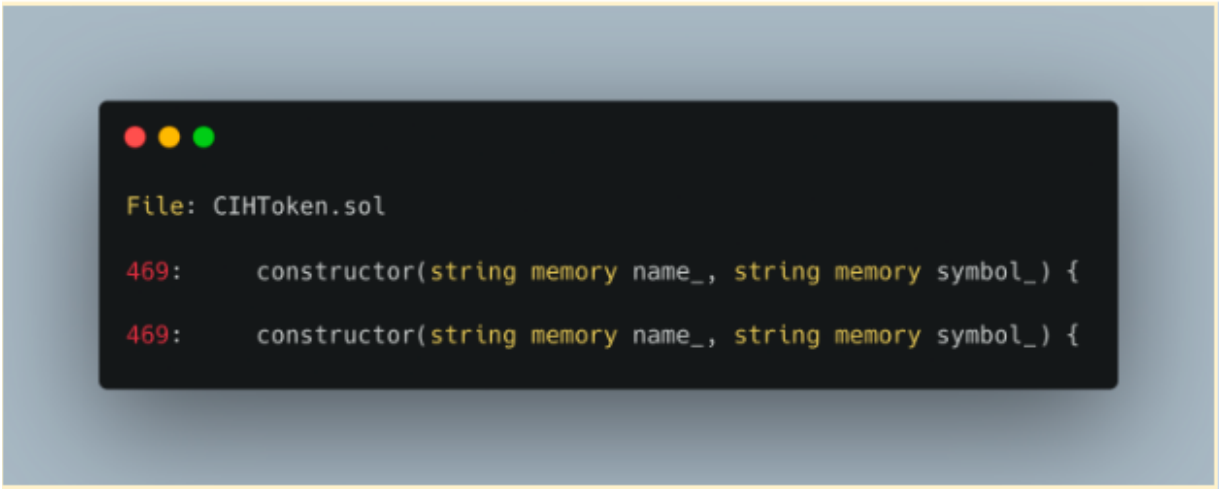
CIH08

Type	Use calldata instead of memory for function arguments that do not get mutated
Severity	Gas Optimisation
File	CIHTOKEN.sol
Instances	2
Status	**Reported**

Description

Mark data types as calldata instead of memory where possible. This makes it so that the data is not automatically loaded into memory. If the data passed into the function does not need to be changed (like updating values in an array), it can be passed in as calldata. The one exception to this is if the argument must later be passed into another function that takes an argument that specifies memory storage.

Snapshot



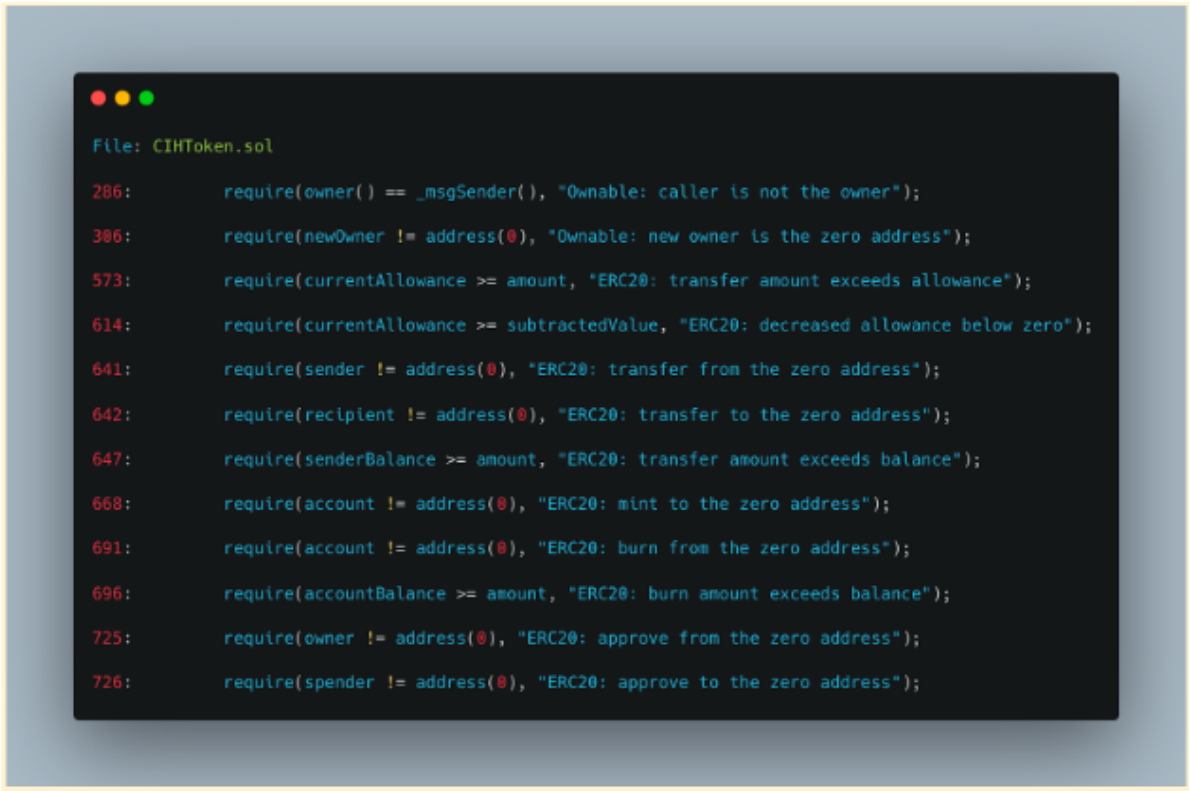
CIH09

Type	Use Custom Errors
Severity	Gas Optimisation
File	CIHTOKEN.sol
Instances	12
Status	**Reported**

Description

Description - Source Instead of using error strings, to reduce deployment and runtime cost, you should use Custom Errors. This would save both deployment and runtime cost.

Snapshot



```
File: CIHToken.sol
286:        require(owner() == _msgSender(), "Ownable: caller is not the owner");
306:        require(newOwner != address(0), "Ownable: new owner is the zero address");
573:        require(currentAllowance >= amount, "ERC20: transfer amount exceeds allowance");
614:        require(currentAllowance >= subtractedValue, "ERC20: decreased allowance below zero");
641:        require(sender != address(0), "ERC20: transfer from the zero address");
642:        require(recipient != address(0), "ERC20: transfer to the zero address");
647:        require(senderBalance >= amount, "ERC20: transfer amount exceeds balance");
668:        require(account != address(0), "ERC20: mint to the zero address");
691:        require(account != address(0), "ERC20: burn from the zero address");
696:        require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
725:        require(owner != address(0), "ERC20: approve from the zero address");
726:        require(spender != address(0), "ERC20: approve to the zero address");
```

CIH10

Type	Long revert strings
Severity	Gas Optimisation
File	CIHTOKEN.sol
Instances	10
Status	**Reported**

Description

Long revert strings

Snapshot

```
File: CIHToken.sol
306:        require(newOwner != address(0), "Ownable: new owner is the zero address");
573:        require(currentAllowance >= amount, "ERC20: transfer amount exceeds allowance");
614:        require(currentAllowance >= subtractedValue, "ERC20: decreased allowance below zero");
641:        require(sender != address(0), "ERC20: transfer from the zero address");
642:        require(recipient != address(0), "ERC20: transfer to the zero address");
647:        require(senderBalance >= amount, "ERC20: transfer amount exceeds balance");
691:        require(account != address(0), "ERC20: burn from the zero address");
696:        require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
725:        require(owner != address(0), "ERC20: approve from the zero address");
726:        require(spender != address(0), "ERC20: approve to the zero address");
```

CIH11

Type	Functions guaranteed to revert when called by normal users can be marked payable
Severity	Gas Optimisation
File	CIHTOKEN.sol
Instances	2
Status	**Reported**

Description

If a function modifier such as onlyOwner is used, the function will revert if a normal user tries to pay the function. Marking the function as payable will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided.

Snapshot

CIH12

Type	Use != 0 instead of > 0 for unsigned integer comparison
Severity	Gas Optimisation
File	CIHTOKEN.sol
Instances	2
Status	**Reported**

Description

Use != 0 instead of > 0 for unsigned integer comparison

Snapshot

CIH13

Type	internal functions not called by the contract should be removed
Severity	Gas Optimisation
File	CIHTOKEN.sol
Instances	13
Status	**Reported**

Description

If the functions are required by an interface, the contract should inherit from that interface and use the override keyword

Snapshot

```
File: CIHToken.sol
43:     function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
56:     function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
68:     function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
85:     function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
97:     function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
114:     function add(uint256 a, uint256 b) internal pure returns (uint256) {
128:     function sub(uint256 a, uint256 b) internal pure returns (uint256) {
142:     function mul(uint256 a, uint256 b) internal pure returns (uint256) {
156:     function div(uint256 a, uint256 b) internal pure returns (uint256) {
172:     function mod(uint256 a, uint256 b) internal pure returns (uint256) {
189:     function sub(
212:     function div(
238:     function mod(
```

APPENDIX

Auditing Approach and Methodologies applied

The Block Audit Report team has performed rigorous testing of the project including the analysis of the code design patterns where we reviewed the smart contract architecture to ensure it is structured along with the safe use of standard inherited contracts and libraries. Our team also conducted a formal line by line inspection of the Smart Contract i.e., a manual review, to find potential issues including but not limited to

- Race conditions
- Zero race conditions approval attacks
- Re-entrancy
- Transaction-ordering dependence
- Timestamp dependence
- Check-effects-interaction pattern (optimistic accounting)
- Decentralized denial-of-service attacks
- Secure ether transfer pattern
- Guard check pattern
- Fail-safe mode
- Gas-limits and infinite loops
- Call Stack depth

In the Unit testing Phase, we coded/conducted custom unit tests written against each function in the contract to verify the claimed functionality from our client. In Automated Testing, we tested the Smart Contract with our standard set of multifunctional tools to identify vulnerabilities and security flaws. The code was tested in collaboration of our multiple team members and this included but not limited to;

- Testing the functionality of the Smart Contract to determine proper logic has been followed throughout the whole process.
- Analyzing the complexity of the code in depth and in detail line-by-line manual review of the code.
- Deploying the code on testnet using multiple clients to run live tests.
- Analyzing failure preparations to check how the Smart Contract performs in case of any bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analyzing the security of the on-chain data.

Issue Categories:

Every issue in this report was assigned a severity level from the following:

Critical Severity Issues

Issues of Critical Severity leaves smart contracts vulnerable to major exploits and can lead to asset loss and data loss. These can have significant impact on the functionality/performance of the smart contract.
We recommend these issues must be fixed before proceeding to MainNet..

High Severity Issues

Issues of High Severity are not as easy to exploit but they might endanger the execution of the smart contract and potentially create crucial problems.
Fixing these issues is highly recommended before proceeding to MainNet.

Medium Severity Issues

Issues on this level are not a major cause of vulnerability to the smart contract, they cannot lead to data-manipulations or asset loss but may affect funtionality.
It is important to fix these issues before proceeding to MainNet.

Low Severity Issues

Issues at this level are very low in their impact on the overall functionality and execution of the smart contract. These are mostly code-level violations or improper formatting.
These issues can be remain unfixed or can be fixed at a later date if the code is redeployed or forked.

Informational Findings

These are finding that our team comes accross when manually reviewing a smart contract which are important to know for the owners as well as users of a contract.
These issues must be acknowledged by the owners before we publish our report.

Ownership Priviledges

Owner of a smart contract can include certain rights and priviledges while deploying a smart contract that might be hidden deep inside the codebase and may make the project vulnerable to rug-pulls or other types of scams.
We at BlockAudit believe in transparency and hence we showcase Ownership priviledges separately so the owner as well as the investors can get a better understanding about the project.

Gas Optimization

Solidity gas optimization is the process of lowering the cost of operating your Solidity smart code. The term "gas" refers to the level of processing power required to perform specific tasks on the Ethereum network.
Each Ethereum transaction costs a fee since it requires the use of computer resources. It will deduct a fee anytime any function in the smart contract is invoked by the contract's owner or users.

DISCLAIMER

This is a limited report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for the client to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that the client should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for the client to conduct the client's own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

By reading this report or any part of it, the client agrees to the terms of this disclaimer. If the client does not agree to the terms, then please immediately cease reading this report, and delete and destroy any/all copies of this report downloaded and/or printed by the client. This report is provided for information purposes only and stays on a non-reliance basis, and does not constitute investment advice. No one/ NONE shall have any rights to rely on the report or its contents, and BlockAudit and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives).

(BlockAudit) owes no duty of care towards the client or any other person, nor does BlockAudit claim any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and BlockAudit hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effects in relation to the report.

Except and only to the extent that it is prohibited by law, BlockAudit hereby excludes all liability and responsibility, and neither the client nor any other person shall have any claim against BlockAudit, for any amount or kind of loss or damage that may result to the client or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report.

The analysis of the security is purely based on the received smart contracts alone. No related/third-party smart contracts, applications or operations were reviewed for security. No product code has been reviewed.

Note: The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the **CHAINHAPPY (CIH)** team put a bug bounty program in place to encourage further analysis of the smart contracts by other third parties

About BlockAudit

BlockAudit is an industry leading security organisation that helps web3 blockchain based projects with their security and correctness of their smart-contracts. With years of experience we have a dedicated team that is capable of performing audits in a wide variety of languages including HTML, PHP, JS, Node, React, Native, Solidity, Rust and other Web3 frameworks for DApps, DeFi, GameFi and Metaverse platforms.

With a mission to make web3 a safe and secure place BlockAudit is commited to provide it's partners with a budget and investor friendly security Audit Report that will increase the value of their projects significantly.

 *www.blockaudit.report*

 *team@blockaudit.report*

 *@BlockAudit*

 *github.com/Block-Audit-Report*