SECURITY ASSESSMENT

HealthyOcean TOKEN

February 15, 2024

Audit Status: Pass

RISK ANALYSIS | HealthyOcean.

Classifications of Manual Risk Results

Classification	Description
● Critical	Danger or Potential Problems.
● High	Be Careful or Fail test.
● Medium	Improve is needed.
● Low	Pass, Not-Detected or Safe Item.
ⓘ Informational	Function Detected

Manual Code Review Risk Results

Contract Security	Description
● Buy Tax	5%
● Sale Tax	5%
● Cannot Buy	Pass
● Cannot Sale	Pass
● Max Tax	10%
ⓘ Modify Tax	No
● Fee Check	Pass
● Is Honeypot?	Not Detected
● Trading Cooldown	Not Detected
● Enable Trade?	False
● Pause Transfer?	Detected

Contract Security	Description
● Max Tx?	Pass
● Is Anti Whale?	Not Detected
● Is Anti Bot?	Not Detected
● Is Blacklist?	Not Detected
● Blacklist Check	Pass
● is Whitelist?	Pass
● Can Mint?	Pass
● Is Proxy?	Not Detected
● Can Take Ownership?	Not Detected
● Hidden Owner?	Not Detected
ⓘ Owner	0x5b33e897501EAC2FeD8B4e904d65D0F6F2e54D69
● Self Destruct?	Not Detected
● External Call?	Detected
● Other?	Not Detected
● Holders	1
● Audit Confidence	Medium
● Authority Check	Pass
● Freeze Check	Pass

The summary section reveals the strengths and weaknesses identified during the assessment, including any vulnerabilities or potential risks that may exist. It serves as a valuable snapshot of the overall security status of the audited project. However, it is highly recommended to read the entire security assessment report for a comprehensive understanding of the findings. The full report provides detailed insights into the assessment process, methodology, and specific recommendations for addressing the identified issues.

CFG Ninja Verified on February 15, 2024

HealthyOcean

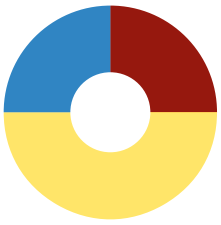
Executive Summary

TYPES	ECOSYSTEM	LANGUAGE
DeFi	BNBCHAIN	Solidity

Timeline

Audit Request	Onboarding Process	Audit Preview	Audit Release
2024-02-02	2024-02-10	2024-02-10	2024-02-11

Vulnerability Summary



4	0	4	4
Total Findings	Resolved	Pending	Unresolved

● 1 Critical	0 Resolved, 1 Pending	Critical risks are the most severe and can have a significant impact on the smart contracts functionality, security, or the entire system. These vulnerabilities can lead to the loss of user funds, unauthorized access, or complete system compromise.
● 0 High		High-risk vulnerabilities have the potential to cause significant harm to the smart contract or the system. While not as severe as critical risks, they can still result in financial losses, data breaches, or denial of service attacks.
● 0 Medium		Medium-risk vulnerabilities pose a moderate level of risk to the smart contracts security and functionality. They may not have an immediate and severe impact but can still lead to potential issues if exploited. These risks should be addressed to ensure the contracts overall security.
● 2 Low	0 Resolved, 2 Pending	Low-risk vulnerabilities have a minimal impact on the smart contracts security and functionality. They may not pose a significant threat, but it is still advisable to address them to maintain a robust security posture.
● 1 Informational	0 Resolved, 1 Pending	Informational risks are not actual vulnerabilities but provide useful information about potential improvements or best practices. These findings may include suggestions for code optimizations, documentation enhancements, or other non-critical areas for improvement.

Token Distribution

Fairlaunch

Tokens allocated for the sale.

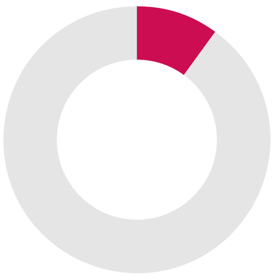
0%

Team and Advisors

Teams

0%

Total Unlock Progress



🟩 Unlocked	0	0%
🟥 Total Locked	100000000	10%
⬜ Untracked	900000000	90%

PROJECT OVERVIEW | HealthyOcean.

Token Summary

Parameter	Result
Address	0x9f8a2aeA53cE5F92964593746F74Bb0E4d958285
Name	HealthyOcean
Token Tracker	HealthyOcean (HLO)
Decimals	9
Supply	1,000,000,000
Platform	BNBCHAIN
Compiler	v0.8.19+commit.7dd6d404
Contract Name	HealthyOcean
Optimization	Yes with 200 runs
LicenseType	MIT
Language	Solidity
Codebase	https://bscscan.com/address/0x9f8a2aea53ce5f92964593746f74bb0e4d958285#code

Main Contract Assessed

Name	Contract	Live
HealthyOcean	0x9f8a2aeA53cE5F92964593746F74Bb0E4d958285	Yes

TestNet Contract Assessed

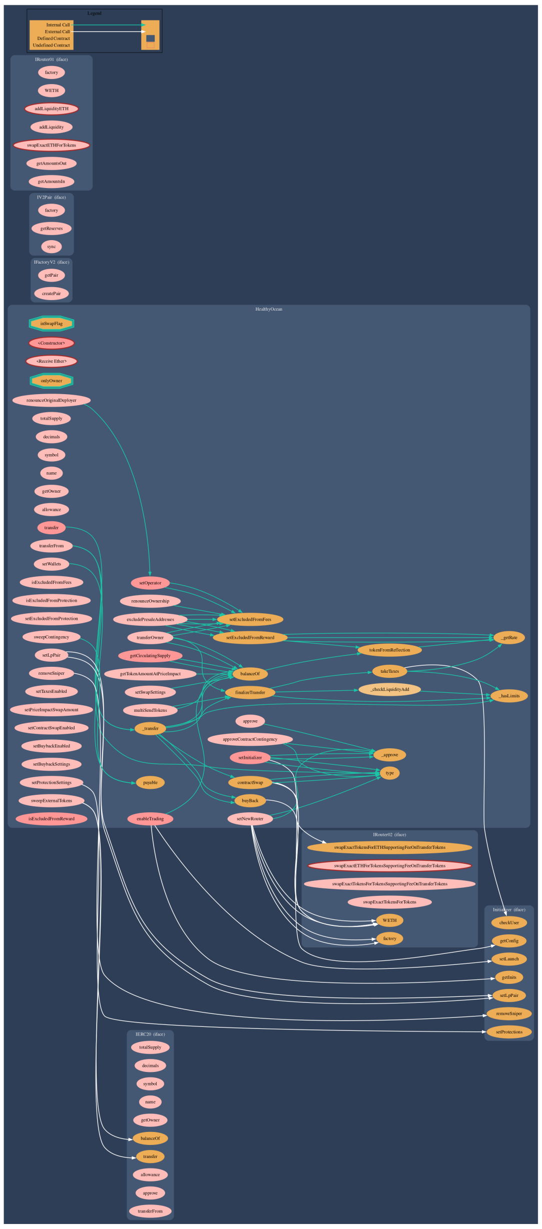
Name	Contract	Live
HealthyOcean	0x6b2bb74Fc6Eff726d5eCa1D81550e7ADe0e428f5	Yes

Solidity Code Provided

SolID	File Sha-1	FileName
HealthyOcean	26d21cf7d07fb9188a50efcb3135ea40820e9e3c	HLO.sol

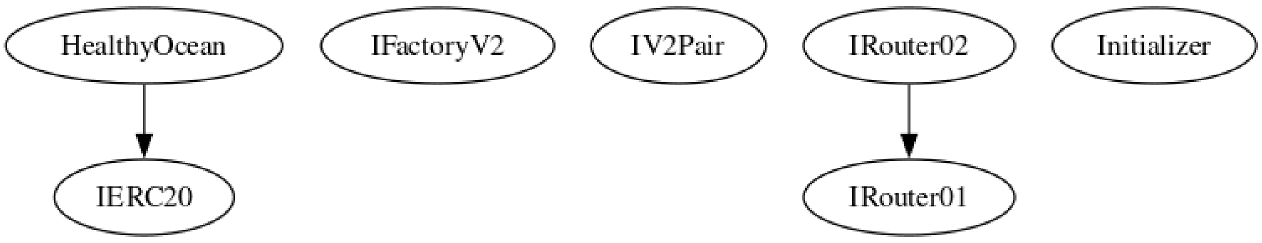
Call Graph

The Smart Contract Graph is a visual representation of the interconnectedness and relationships between smart contracts within a blockchain network. It provides a comprehensive view of the interactions and dependencies between different smart contracts, allowing developers and users to analyze and understand the flow of data and transactions within the network. The Smart Contract Graph enables better transparency, security, and efficiency in decentralized applications by facilitating the identification of potential vulnerabilities, optimizing contract execution, and enhancing overall network performance.



Inheritance Check

Smart contract inheritance is a concept in blockchain programming where one smart contract can inherit properties and functionalities from another existing smart contract. This allows for code reuse and modularity, making the development process more efficient and scalable. Inheritance enables the child contract to access and utilize the variables, functions, and modifiers defined in the parent contract, thereby inheriting its behavior and characteristics. This feature is particularly useful in complex decentralized applications (dApps) where multiple contracts need to interact and share common functionalities. By leveraging smart contract inheritance, developers can create more organized and maintainable code structures, promoting code reusability and reducing redundancy.



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 | HealthyOcean.

SWC-108 - State Variable Default Visibility.

CWE-710: Improper Adherence to Coding Standards

Description:

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

Remediation:

Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables.

References:

Ethereum Smart Contract Best Practices - Explicitly mark visibility in functions and state variables

Smart Contract Vulnerability Details | SWC-115 - Authorization through tx.origin.

CWE-477: Use of Obsolete Function

Description:

tx.origin is a global variable in Solidity which returns the address of the account that sent the transaction. Using the variable for authorization could make a contract vulnerable if an authorized account calls into a malicious contract. A call could be made to the vulnerable contract that passes the authorization check since tx.origin returns the original sender of the transaction which in this case is the authorized account.

Remediation:

tx.origin should not be used for authorization. Use msg.sender instead.

References:

Solidity Documentation - tx.origin

Ethereum Smart Contract Best Practices - Avoid using tx.origin

SigmaPrime - Visibility.

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.

Smart Contract Vulnerability Details | SWC-123 - Requirement Violation.

CWE-573: Improper Following of Specification by Caller

Description:

The Solidity require() construct is meant to validate external inputs of a function. In most cases, such external inputs are provided by callers, but they may also be returned by callees. In the former case, we refer to them as precondition violations. Violations of a requirement can indicate one of two possible issues:

A bug exists in the contract that provided the external input.
The condition used to express the requirement is too strong.

Remediation:

If the required logical condition is too strong, it should be weakened to allow all valid external inputs.Otherwise, the bug must be in the contract that provided the external input and one should consider fixing its code by making sure no invalid inputs are provided.

References:

The use of revert(), assert(), and require() in Solidity, and the new REVERT opcode in the EVM

TECHNICAL FINDINGS | HealthyOcean.

Smart contract security audits classify risks into several categories: Critical, High, Medium, Low, and Informational. These classifications help assess the severity and potential impact of vulnerabilities found in smart contracts.

Classification of Risk

Severity	Description
● Critical	Critical risks are the most severe and can have a significant impact on the smart contracts functionality, security, or the entire system. These vulnerabilities can lead to the loss of user funds, unauthorized access, or complete system compromise.
● High	High-risk vulnerabilities have the potential to cause significant harm to the smart contract or the system. While not as severe as critical risks, they can still result in financial losses, data breaches, or denial of service attacks.
● Medium	Medium-risk vulnerabilities pose a moderate level of risk to the smart contracts security and functionality. They may not have an immediate and severe impact but can still lead to potential issues if exploited. These risks should be addressed to ensure the contracts overall security.
● Low	Low-risk vulnerabilities have a minimal impact on the smart contracts security and functionality. They may not pose a significant threat, but it is still advisable to address them to maintain a robust security posture.
ⓘ Informational	Informational risks are not actual vulnerabilities but provide useful information about potential improvements or best practices. These findings may include suggestions for code optimizations, documentation enhancements, or other non-critical areas for improvement.

By categorizing risks into these classifications, smart contract security audits can prioritize the resolution of critical and high-risk vulnerabilities to ensure the contract's overall security and protect user funds and data.

HLO-02 | Function Visibility Optimization.

Category	Severity	Location	Status
Gas Optimization	● Informational	HLO.sol: L: 350 C: 14, L: 368 C: 14, L: 586 C: 14, L: 627 C: 14	Detected

Description

The following functions are declared as public and are not invoked in any of the contracts contained within the projects scope:

Function Name	Parameters	Visibility
setInitializer	address init	Public
setExcludedFromFees	address account, bool enabled	Public
enableTrading		Publice
setExcludedFromReward	address account, bool enabled	Public

The functions that are never called internally within the contract should have external visibility

Recommendation

We advise that the function's visibility specifiers are set to external, and the array-based arguments change their data location from memory to calldata, optimizing the gas cost of the function.

Mitigation

References:

external vs public best practices.

HLO-03 | Lack of Input Validation.

Category	Severity	Location	Status
Volatile Code	🟢 Low	HLO.sol: L: 306 C: 14, L: 368 C: 14, L: 372 C: 14, L: 380 C: 14, L: 384 C: 14, L: 388 C: 14, L: 392 C: 14, L: 420 C: 14, L: 426 C: 14, L: 430 C: 14	Detected

Description

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

The given input is missing the check for the onlyOwners need to be revisited for require..

Recommendation

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. onlyOwners need to be revisited for require..

Mitigation

References:

Zero Address check. The danger!!!

HLO-05 | Missing Event Emission.

Category	Severity	Location	Status
Volatile Code	🟢 Low	HLO.sol: L: 306 C: 14, L: 319 C: 14, L: 335 C: 14, L: 350 C: 14, L: 368 C: 14, L: 372 C: 14, L: 380 C: 14, L: 384 C: 14, L: 388 C: 14, L: 392 C: 14, L: 406 C: 14, L: 415 C: 14, L: 426 C: 14, L: 430 C: 14, L: 435 C: 14, L: 586 C: 14, L: 602 C: 14, L: 607 C: 14, L: 615 C: 14, L: 627 C: 14	Detected

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.

Recommendation

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

Mitigation

References:

Understanding Events in Smart Contracts

HLO-18 | Stop Transactions by using Enable Trade.

Category	Severity	Location	Status
Logical Issue	● Critical	HLO.sol: L: 586 C: 14, L: 176 C: 17	Detected

Description

Enable Trade is present on the following contract and when combined with Exclude from fees it can be considered a whitelist process, this will allow anyone to trade before others and can represent and issue for the holders.

Recommendation

We recommend the project owner to carefully review this function and avoid problems when performing both actions.

Mitigation

References:

Writing Clean Code for Solidity: Best Practices for Solidity Development

FINDINGS

In this document, we present the findings and results of the smart contract security audit. The identified vulnerabilities, weaknesses, and potential risks are outlined, along with recommendations for mitigating these issues. It is crucial for the team to address these findings promptly to enhance the security and trustworthiness of the smart contract code.

Severity	Found	Pending	Resolved
● Critical	1	1	0
● High	0	0	0
● Medium	0	0	0
● Low	2	2	0
● Informational	1	1	0
Total	4	4	0

In a smart contract, a technical finding summary refers to a compilation of identified issues or vulnerabilities discovered during a security audit. These findings can range from coding errors and logical flaws to potential security risks. It is crucial for the project owner to thoroughly review each identified item and take necessary actions to resolve them. By carefully examining the technical finding summary, the project owner can gain insights into the weaknesses or potential threats present in the smart contract. They should prioritize addressing these issues promptly to mitigate any risks associated with the contract's security. Neglecting to address any identified item in the security audit can expose the smart contract to significant risks. Unresolved vulnerabilities can be exploited by malicious actors, potentially leading to financial losses, data breaches, or other detrimental consequences. To ensure the integrity and security of the smart contract, the project owner should engage in a comprehensive review process. This involves understanding the nature and severity of each identified item, consulting with experts if needed, and implementing appropriate fixes or enhancements. Regularly updating and maintaining the smart contract's codebase is also essential to address any emerging security concerns. By diligently reviewing and resolving all identified items in the technical finding summary, the project owner can significantly reduce the risks associated with the smart contract and enhance its overall security posture.

SOCIAL MEDIA CHECKS | HealthyOcean.

Social Media	URL	Result
Website	https://healthyocean.finance/	Pass
Telegram	https://t.me/HealthyOceanBSC	Pass
Twitter	https://twitter.com/HealthyOceanBSC	Pass
Facebook		N/A
Reddit	N/A	N/A
Instagram	N/A	N/A
CoinGecko	N/A	N/A
Github		N/A
CMC	N/A	N/A
Email	N/A	Contact
Other	https://www.youtube.com/@HealthyOceanBSC	Pass

From a security assessment standpoint, inspecting a project's social media presence is essential. It enables the evaluation of the project's reputation, credibility, and trustworthiness within the community. By analyzing the content shared, engagement levels, and the response to any security-related incidents, one can assess the project's commitment to security practices and its ability to handle potential threats.

Social Media Information Notes:

Auditor Notes: Website needs a bit of improvement.

Project Owner Notes:

ASSESSMENT RESULTS | HealthyOcean.

Score Rsesults

Review	Score
Overall Score	81/100
Auditor Score	80/100

Review by Section	Score
Manual Scan Score	49
SWC Scan Score	27
Advance Check Score	5

Our security assessment or audit score system for the smart contract and project follows a comprehensive evaluation process to ensure the highest level of security. The system assigns a score based on various security parameters and benchmarks, with a passing score set at 80 out of a total attainable score of 100.The assessment process includes a thorough review of the smart contracts codebase, architecture, and design principles. It examines potential vulnerabilities, such as code bugs, logical flaws, and potential attack vectors. The evaluation also considers the adherence to best practices and industry standards for secure coding. Additionally, the system assesses the projects overall security measures, including infrastructure security, data protection, and access controls. It evaluates the implementation of encryption, authentication mechanisms, and secure communication protocols. To achieve a passing score, the smart contract and project must attain a minimum of 80 points out of the total attainable score of 100. This ensures that the system has undergone a rigorous security assessment and meets the required standards for secure operation.

Important Notes for HLO

- Only a few issues/vulnerabilities were found.
- The SAFU Dev needs to enable trade.
- A few functions are missing require and emit.
- Contract by Trynos.

Auditor Score =80
Audit Passed

Appendix

Finding Categories

Centralization / Privilege

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

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimalEVM 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 howblock.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functionsbeing 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 mayresult in a vulnerability.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to makethe 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 require statements on the input variables than a setterfunction.

Coding Best Practices

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

Disclaimer

The purpose of this disclaimer is to outline the responsibilities and limitations of the security assessment and smart contract audit conducted by Bladepool/CFG NINJA. By engaging our services, the project owner acknowledges and agrees to the following terms:

1. Limitation of Liability: Bladepool/CFG NINJA shall not be held liable for any damages, losses, or expenses incurred as a result of any contract malfunctions, vulnerabilities, or exploits discovered during the security assessment and smart contract audit. The project owner assumes full responsibility for any consequences arising from the use or implementation of the audited smart contract. 2. No Guarantee of Absolute Security: While Bladepool/CFG NINJA employs industry-standard practices and methodologies to identify potential security risks, it is important to note that no security assessment or smart contract audit can provide an absolute guarantee of security. The project owner acknowledges that there may still be unknown vulnerabilities or risks that are beyond the scope of our assessment. 3. Transfer of Responsibility: By engaging our services, the project owner agrees to assume full responsibility for addressing and mitigating any identified vulnerabilities or risks discovered during the security assessment and smart contract audit. It is the project owner s sole responsibility to ensure the proper implementation of necessary security measures and to address any identified issues promptly. 4. Compliance with Applicable Laws and Regulations: The project owner acknowledges and agrees to comply with all applicable laws, regulations, and industry standards related to the use and implementation of smart contracts. Bladepool/CFG NINJA shall not be held responsible for any non-compliance by the project owner. 5. Third-Party Services: The security assessment and smart contract audit conducted by Bladepool/CFG NINJA may involve the use of third-party tools, services, or technologies. While we exercise due diligence in selecting and utilizing these resources, we cannot be held liable for any issues or damages arising from the use of such third-party services. 6. Confidentiality: Bladepool/CFG NINJA maintains strict confidentiality regarding all information and data obtained during the security assessment and smart contract audit. However, we cannot guarantee the security of data transmitted over the internet or through any other means. 7. Not a Financial Advice: Bladepool/CFG NINJA please note that the information provided in the security assessment or audit should not be considered as financial advice. It is always recommended to consult with a financial professional or do thorough research before making any investment decisions.

By engaging our services, the project owner acknowledges and accepts these terms and releases Bladepool/CFG NINJA from any liability, claims, or damages arising from the security assessment and smart contract audit. It is recommended that the project owner consult legal counsel before entering into any agreement or contract.