CFG NINJA

SECURITY ASSESSMENT

Chi Yamada Cat TOKEN

April 13, 2024

Audit Status: Fail

BLADE POOL

RISK ANALYSIS | Chi Yamada Cat.

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	7%
Sale Tax	7%
Cannot Buy	Pass
Cannot Sale	Pass
Max Tax	7%
Modify Tax	No
Fee Check	Pass
Is Honeypot?	Detected
Trading Cooldown	Not Detected
Enable Trade?	false
Pause Transfer?	Not Detected

Contract Security	Description
Max Tx?	Pass
Is Anti Whale?	Detected
Is Anti Bot?	Not Detected
Is Blacklist?	Not Detected
Blacklist Check	Pass
is Whitelist?	Not Detected
Can Mint?	Pass
Is Proxy?	Not Detected
Can Take Ownership?	Not Detected
Hidden Owner?	Not Detected
Owner	0x8C49A61F7Fe943da6d921719DFc4ABb56c2b8877
Self Destruct?	Not Detected
External Call?	Not Detected
Other?	Not Detected
Holders	3
Audit Confidence	Very Low
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 April 13, 2024

Chi Yamada Cat

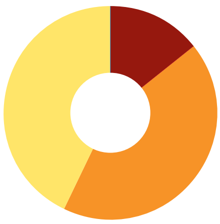
Executive Summary

TYPES	ECOSYSTEM	LANGUAGE
DeFi	BNBCHAIN	Solidity

Timeline

Audit Request	Onboarding Process	Audit Preview	Audit Release
2024-04-13	2024-04-13	2024-04-13	2024-04-13

Vulnerability Summary



Total Findings	0 Resolved	9 Pending	0 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.
3 Medium	0 Resolved, 3 Pending	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.
3 Low	0 Resolved, 3 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.
0 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.

PROJECT OVERVIEW | Chi Yamada Cat.

Token Summary

Parameter	Result
Address	0x4365a0B158575d6a50196A0df873BB81b90192dF
Name	Chi Yamada Cat
Token Tracker	Chi Yamada Cat (CYCat)
Decimals	18
Supply	420,960,000,000,000
Platform	BNBCHAIN
Compiler	v0.8.18+commit.87f61d96
Contract Name	CYCat
Optimization	Yes with 200 runs
LicenseType	MIT
Language	Solidity
Codebase	https://bscscan.com/token/0x4365a0b158575d6a50196a0df873bb81b90192df#code

Main Contract Assessed

Name	Contract	Live
Chi Yamada Cat	0x4365a0B158575d6a50196A0df873BB81b90192dF	Yes

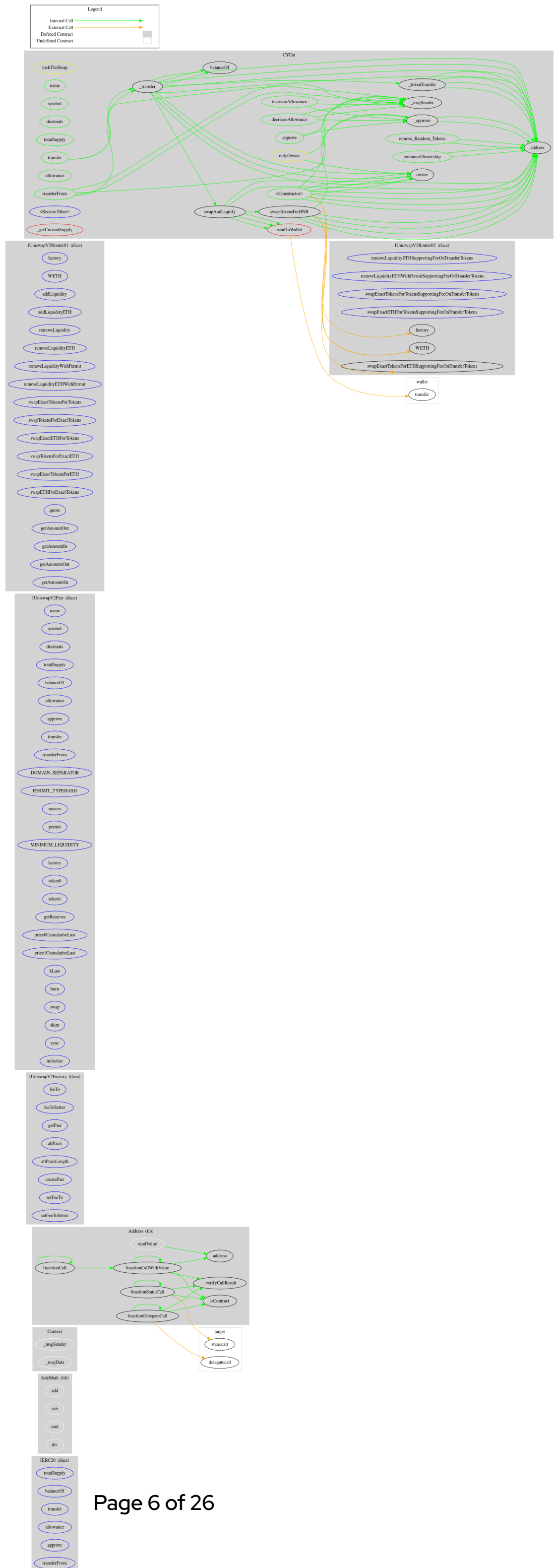
TestNet Contract Was Not Assessed

Solidity Code Provided

SolID	File Sha-1	FileName
CYCat	607bed14b4afc0e3eecbfa67497cdb65fff8c368	CYCat.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.



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	CYCat.sol	L: 0 C: 0
SWC-101	Pass	Integer Overflow and Underflow.	CYCat.sol	L: 0 C: 0
SWC-102	Pass	Outdated Compiler Version file.	CYCat.sol	L: 0 C: 0
SWC-103	Low	A floating pragma is set.	CYCat.sol	L: 12 C: 0
SWC-104	Pass	Unchecked Call Return Value.	CYCat.sol	L: 0 C: 0
SWC-105	Pass	Unprotected Ether Withdrawal.	CYCat.sol	L: 0 C: 0
SWC-106	Pass	Unprotected SELFDESTRUCT Instruction	CYCat.sol	L: 0 C: 0
SWC-107	Pass	Read of persistent state following external call.	CYCat.sol	L: 0 C: 0
SWC-108	Pass	State variable visibility is not set..	CYCat.sol	L: 0 C: 0
SWC-109	Pass	Uninitialized Storage Pointer.	CYCat.sol	L: 0 C: 0
SWC-110	Pass	Assert Violation.	CYCat.sol	L: 0 C: 0
SWC-111	Pass	Use of Deprecated Solidity Functions.	CYCat.sol	L: 0 C: 0
SWC-112	Pass	Delegate Call to Untrusted Callee.	CYCat.sol	L: 0 C: 0
SWC-113	Pass	Multiple calls are executed in the same transaction.	CYCat.sol	L: 0 C: 0
SWC-114	Pass	Transaction Order Dependence.	CYCat.sol	L: 0 C: 0
SWC-115	Low	Authorization through tx.origin.	CYCat.sol	L: 857 C: 57,L: 861 C: 53
SWC-116	Pass	A control flow decision is made based on The block.timestamp environment variable.	CYCat.sol	L: 0 C: 0
SWC-117	Pass	Signature Malleability.	CYCat.sol	L: 0 C: 0

ID	Severity	Name	File	location
SWC-118	Pass	Incorrect Constructor Name.	CYCat.sol	L: 0 C: 0
SWC-119	Pass	Shadowing State Variables.	CYCat.sol	L: 0 C: 0
SWC-120	Low	Potential use of block.number as source of randonmness.	CYCat.sol	L: 858 C: 32, L: 861 C: 66
SWC-121	Pass	Missing Protection against Signature Replay Attacks.	CYCat.sol	L: 0 C: 0
SWC-122	Pass	Lack of Proper Signature Verification.	CYCat.sol	L: 0 C: 0
SWC-123	Pass	Requirement Violation.	CYCat.sol	L: 0 C: 0
SWC-124	Pass	Write to Arbitrary Storage Location.	CYCat.sol	L: 0 C: 0
SWC-125	Pass	Incorrect Inheritance Order.	CYCat.sol	L: 0 C: 0
SWC-126	Pass	Insufficient Gas Griefing.	CYCat.sol	L: 0 C: 0
SWC-127	Pass	Arbitrary Jump with Function Type Variable.	CYCat.sol	L: 0 C: 0
SWC-128	Pass	DoS With Block Gas Limit.	CYCat.sol	L: 0 C: 0
SWC-129	Pass	Typographical Error.	CYCat.sol	L: 0 C: 0
SWC-130	Pass	Right-To-Left-Override control character (U+202E).	CYCat.sol	L: 0 C: 0
SWC-131	Pass	Presence of unused variables.	CYCat.sol	L: 0 C: 0
SWC-132	Pass	Unexpected Ether balance.	CYCat.sol	L: 0 C: 0
SWC-133	Pass	Hash Collisions with Multiple Variable Length Arguments.	CYCat.sol	L: 0 C: 0
SWC-134	Pass	Message call with hardcoded gas amount.	CYCat.sol	L: 0 C: 0
SWC-135	Pass	Code With No Effects (Irrelevant/ Dead Code).	CYCat.sol	L: 0 C: 0
SWC-136	Pass	Unencrypted Private Data On-Chain.	CYCat.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-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.

TECHNICAL FINDINGS | Chi Yamada Cat.

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.

CYCat-01 | Potential Sandwich Attacks.

Category	Severity	Location	Status
Security	Medium	CYCat.sol: L:479, C:14	Detected

Description

A sandwich attack might happen when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by frontrunning (before the transaction being attacked) a transaction to purchase one of the assets and make profits by back running (after the transaction being attacked) a transaction to sell the asset. The following functions are called without setting restrictions on slippage or minimum output amount, so transactions triggering these functions are vulnerable to sandwich attacks, especially when the input amount is large:

- swapExactTokensForETHSupportingFeeOnTransferTokens()

Recommendation

We recommend setting reasonable minimum output amounts, instead of 0, based on token prices when calling the aforementioned functions.

Mitigation

References:

What Are Sandwich Attacks in DeFi — and How Can You Avoid Them?.

CYCat-05 | Missing Event Emission.

Category	Severity	Location	Status
Volatile Code	Low	CYCat.sol: L: 488 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

CYCat-06 | Conformance with Solidity Naming Conventions.

Category	Severity	Location	Status
Coding Style	Low	CYCat.sol: L: 488, C:11, L: 317-320, C:11	Detected

Description

Solidity defines a naming convention that should be followed. Rule exceptions: Allow constant variable name/symbol/decimals to be lowercase. Allow _ at the beginning of the mixed_case match for private variables and unused parameters.

```
remove_Random_Tokens
Wallet_Marketing
Wallet_Ecology
Wallet_Buyback
Wallet_Burn
```

Recommendation

Follow the Solidity naming convention.

Mitigation

References:

https://docs.soliditylang.org/en/v0.4.25/style-guide.html#naming-convention

Writing Clean Code for Solidity: Best Practices for Solidity Development

CYCat-08 | Dead Code Elimination.

Category	Severity	Location	Status
Coding Style	Low	CYCat.sol: L: 299 C: 0	Detected

Description

Functions that are not used in the contract, and make the code s size bigger.

SafeMath

Recommendation

Remove unused functions. dead-code elimination (also known as DCE, dead-code removal, dead-code stripping, or dead-code strip) is a compiler optimization to remove code which does not affect the program results. Removing such code has several benefits: it shrinks program size, an important consideration in some contexts, and it allows the running program to avoid executing irrelevant operations, which reduces its running time. It can also enable further optimizations by simplifying program structure.

Mitigation

References:

Cheatsheetl

Writing Clean Code for Solidity: Best Practices for Solidity Development

CYCat-11 | Missing Fee Exclusion Management Function..

Category	Severity	Location	Status
Functionality	High	CYCat.sol: L: 0 C: 0	Detected

Description

There is no function to dynamically update the _isExcludedFromFee mapping to manage fee exemptions for addresses.

Recommendation

Implement an onlyOwner function to add or remove addresses from fee exclusion.

Mitigation

References:

Writing Clean Code for Solidity: Best Practices for Solidity Development

CYCat-14 | Unnecessary Use Of SafeMath.

Category	Severity	Location	Status
Logical Issue	Medium	CYCat.sol: L: 25 C: 14	Detected

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.

Recommendation

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

Mitigation

References:

Writing Clean Code for Solidity: Best Practices for Solidity Development

CYCat-23 | Unprotected renounceOwnership Function.

Category	Severity	Location	Status
Coding Style	Critical	CYCat.sol: L: 310 C: 14	Detected

Description

The renounceOwnership function can be called by any user, allowing ownership to be renounced unintentionally..

Recommendation

Add the onlyOwner modifier to the renounceOwnership function..

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	3	3	0
Low	3	3	0
Informational	0	0	0
Total	7	9	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 | Chi Yamada Cat.

Social Media	URL	Result
Website	https://cycat.art/	Pass
Telegram	https://t.me/CYCatBSC	Pass
Twitter	https://twitter.com/CYCatBSC	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		Fail

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 | Chi Yamada Cat.

Score Rsesults

Review	Score
Overall Score	52/100
Auditor Score	60/100

Review by Section	Score
Manual Scan Score	6
SWC Scan Score	31
Advance Check Score	15

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 CYCat

- Redundant SafeMath Library: Unnecessary with Solidity 0.8's built-in checks.▌
- Missing Fee Exclusion Management Function: No way to update _isExcludedFromFee.▌
- Potential Fee Handling Issue for Contracts: Contracts may fail to transfer tokens due to fees.▌
- Unprotected renounceOwnership Function: Can be called by any user, posing a risk.▌
- Missing Event Emission in Token Removal Function: Reduces transparency of token removals.▌
- Non-Standard Function Naming: remove_Random_Tokens does not follow mixedCase convention.▌
- State Variables Not Declared Constant: Some variables could be declared constant to save gas.▌
- Non-Standard State Variable Naming: Wallet_Burn does not follow mixedCase convention.▌
- Overall Classification: Needs Improvement▌

- Updated Conclusion: The contract has several areas for improvement, including optimization issues like redundant SafeMath and non-constant state variables, functionality gaps such as missing fee exclusion management, security concerns with unprotected ownership renunciation, and style deviations from Solidity naming conventions. Addressing these issues is crucial for contract security, gas efficiency, and adherence to best practices. A comprehensive audit and testing are recommended before deployment to ensure the contract operates securely and as intended.

Auditor Score =60
Audit Fail

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.

CFG NINJA AUDITS