



SECURITY ASSESSMENT Julius Czer TOKEN

February 12, 2025

Audit Status: Pass














RISK ANALYSIS | Julius Czer.

■ 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	3%
 Sale Tax	3%
 Cannot Buy	Pass
 Cannot Sale	Pass
 Max Tax	3%
 Modify Tax	Yes
 Fee Check	Pass
 Is Honeypot?	Not Detected
 Trading Cooldown	Not Detected
 Enable Trade?	False
 Pause Transfer?	Not-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?	Detected
● Can Mint?	Pass
● Is Proxy?	Not Detected
● Can Take Ownership?	Not Detected
● Hidden Owner?	Not-Detected
i Owner	0xb6B0292B862e93f53388824c53921a9Ded28B1F8
● Self Destruct?	Not Detected
● External Call?	Not-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 12, 2025

**Julius Czer**

Executive Summary

TYPES

DeFi

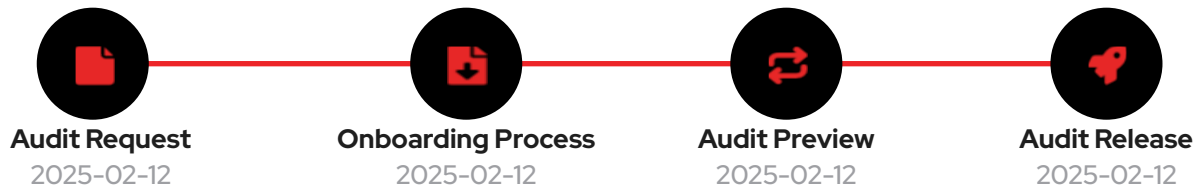
ECOSYSTEM

BNBCHAIN

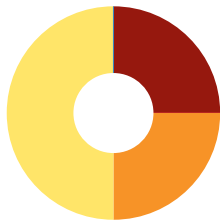
LANGUAGE

Solidity

Timeline



Vulnerability Summary



4
Total Findings

0
Resolved

4
Pending

4
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.

1 Medium

0 Resolved, 1 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.

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.

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 | Julius Czer.

Token Summary

Parameter	Result
Address	0x8F420a7d21870A2025cDF4f7efd66228d97f339B
Name	Julius Czer
Token Tracker	Julius Czer (CZER)
Decimals	18
Supply	420,000,000,000,000,000
Platform	BNBCHAIN
Compiler	v0.8.19+commit.7dd6d404
Contract Name	JuliusCzer
Optimization	Yes with 200 runs
LicenseType	MIT
Language	Solidity
Codebase	https://bscscan.com/address/0x8F420a7d21870A2025cDF4f7efd66228d97f339B#code

■ Main Contract Assessed

Name	Contract	Live
Julius Czer	0x8F420a7d21870A2025cDF4f7efd66228d97f339B	Yes

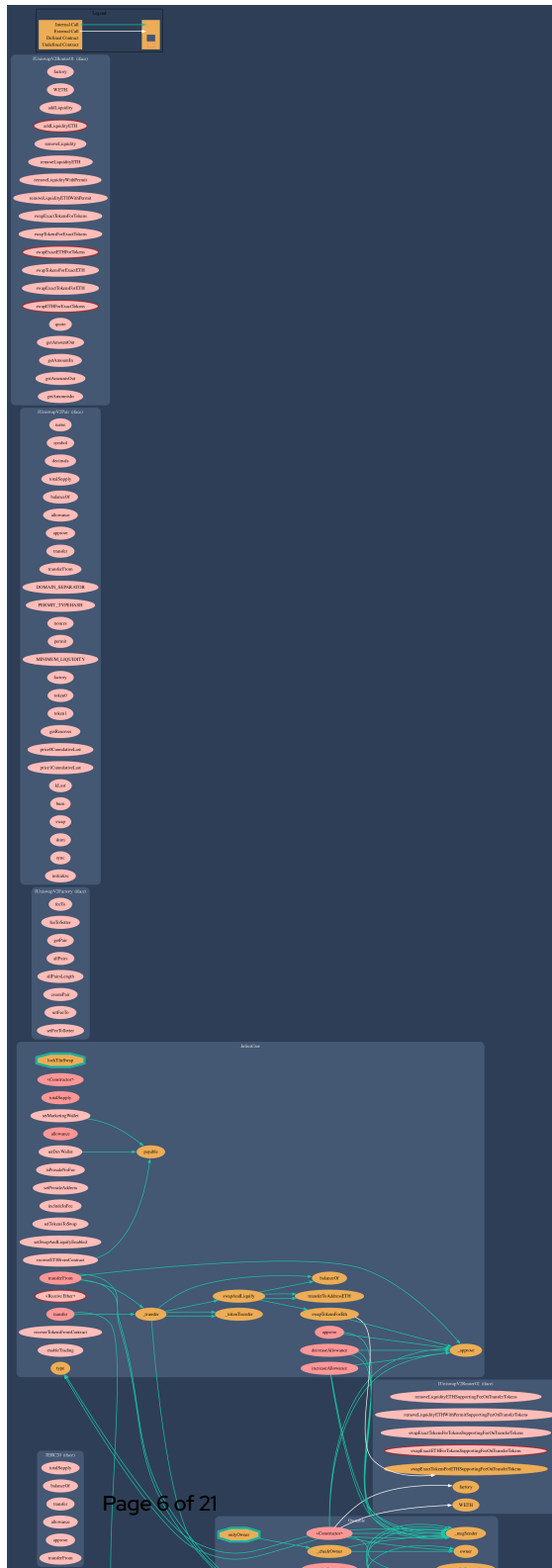
■ TestNet Contract Was Not Assessed

■ Solidity Code Provided

SolidID	File Sha-1	FileName
JuliusCzer	80af70734c98374c63739ded5375c7605eb8d724	JuliusCzer.sol

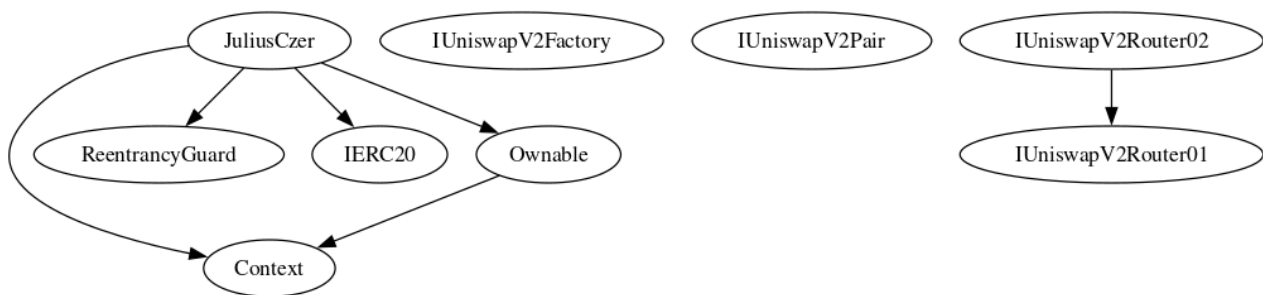
Call Graph

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.



TECHNICAL FINDINGS | Julius Czer.



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.

CZER-03 | Lack of Input Validation.

Category	Severity	Location	Status
Volatile Code	 Low	JuliusCzer.sol: L: 178 C: 12, L: 918 C: 12	 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!!!

CZER-05 | Missing Event Emission.

Category	Severity	Location	Status
Volatile Code	 Low	JuliusCzer.sol: L: 178 C: 12, L: 186 C: 12	 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

CZER-18 | Stop Transactions by using Enable Trade.

Category	Severity	Location	Status
Logical Issue	 Critical	JuliusCzer.sol: L: 1004 C: 14	 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

CZER-19 | Centralization Privileges of CZER.

Category	Severity	Location	Status
Coding Style	● Medium	JuliusCzer.sol: L: 0 C: 14	 Detected

Description

In a smart contract, the concept of "onlyOwner" functions refers to certain functions that can only be executed by the owner or creator of the contract. These functions are typically designed to perform critical actions or modify sensitive data within the contract. By restricting access to these functions, the contract owner maintains control and ensures the integrity and security of the contract.

Function Name	Parameters	Visibility
renounceOwnership		Public
transferOwnership		Public
setPresaleAddress		External
includeInFee		External
setTokensToSwap		External
setSwapAndLiquifyEnabled		External
setMarketingWallet		External
setDevWallet		External
enableTrading		External

Recommendation

Inheriting from Ownable and calling its constructor on yours ensures that the address deploying your contract is registered as the owner. The onlyOwner modifier makes a

function revert if not called by the address registered as the owner. It is important that deployer or owner secure the credentials that has owner privilege to ensure the security of the project.

Mitigation






References:

Guide to Ownership and Access Control in Solidity

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	1	1	0
 Low	2	2	0
 Informational	0	0	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 | Julius Czer.

Social Media	URL	Result
Website	https://juliusczer.io/	Pass
Telegram	https://t.me/JuliusCzerBSC	Pass
Twitter	https://x.com/juliusczbsc	Pass
Facebook		N/A
Reddit		N/A
Instagram		N/A
CoinGecko		N/A
Github		N/A
CMC	N/A	N/A
Email		Contact
Other		N/A

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:

Project Owner Notes:

ASSESSMENT RESULTS | Julius Czer.

Score Results

Review	Score
Overall Score	87/100
Auditor Score	87/100

Review by Section	Score
Manual Scan Score	44
SWC Scan Score	37
Advance Check Score	6

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 CZER

- **Code Structure & Readability:** The code is well-structured with clear comments and event logging. Consider adding more detailed comments for complex functions.■
- **Reentrancy Protection:** Proper use of ReentrancyGuard to prevent reentrant attacks. Ensure no external calls are made before state changes.■
- **Ownership & Access Control:** Ownership functions are protected with onlyOwner. Consider using a multi-signature wallet for critical functions.■
- **Fee Management:** Fees are adjustable by the owner. Ensure transparency to users. Verify that fee calculations are accurate and fair.■
- **Liquidity Functions:** swapAndLiquify manages liquidity effectively. Monitor for potential slippage and ensure sufficient liquidity.■
- **Trading Control:** Trading can be enabled/disabled, which is a central control point. Ensure the owner's intentions are aligned with the community.■
- **Token Recovery:** Functions to recover tokens and ETH

are present. Ensure these are used responsibly to avoid misuse.■

- External Calls: Relies on Uniswap V2 interfaces; ensure they are up-to-date and secure. Test on different networks to ensure compatibility.■
- Gas Optimization: Review for potential gas savings, especially in loops and state changes. Consider optimizing storage and function calls.■
- Testing & Deployment: Conduct thorough testing, especially around edge cases and deployment scenarios.■
- Event Logging: Events are well-implemented for tracking state changes. Ensure all critical actions are logged for transparency.■
- Security Best Practices: Follow Solidity security best practices, such as using require statements for input validation. Regularly update dependencies and libraries to mitigate vulnerabilities.■
- Community & Governance: Consider implementing a governance mechanism for community involvement in decision-making. Transparency in changes to fees and trading status is crucial.■

- Documentation: Ensure comprehensive documentation is available for users and developers. Include details on how fees are calculated and distributed.■
- Future Updates: Plan for future updates and ensure the contract is upgradeable if necessary. Consider implementing a timelock for critical changes to allow community oversight.

Auditor Score =87
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

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.

