CFG NINJA

SECURITY ASSESSMENT

Spiral Token

January 9, 2026

Audit Status: Fail

BLADE POOL

RISK ANALYSIS | Spiral.

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	No Limit (Can be 100%)
Modify Tax	Yes
Fee Check	Fail
Is Honeypot?	Not Detected
Trading Cooldown	Not Detected
Enable Trade?	true
Pause Transfer?	Detected

Contract Security	Description
Max Tx?	Detected
Is Anti Whale?	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?	Detected
Hidden Owner?	Not Detected
Owner	TBD
Self Destruct?	Not Detected
External Call?	Detected
Other?	Uniswap V2 integration, Jackpot mechanism, Buyback system
Holders	1
Audit Confidence	Very High Risk
Authority Check	Fail
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 January 9, 2026

Spiral

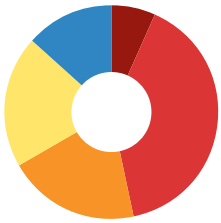
Executive Summary

TYPES	ECOSYSTEM	LANGUAGE
DeFi	Base	Solidity

Timeline

Audit Request	Onboarding Process	Audit Preview	Audit Release
2026-01-08	2026-01-08	2026-01-08	2026-01-08

Vulnerability Summary



13	0	13	13
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.
6 High	0 Resolved, 6 Pending	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.
2 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.

Total Unlock Progress

Unlocked	0	0%
Total Locked	0	0%
Untracked	10000000	100%

PROJECT OVERVIEW | Spiral.

Token Summary

Parameter	Result
Address	0x4
Name	Spiral
Token Tracker	Spiral (SPIRAL)
Decimals	9
Supply	10,000,000
Platform	Base
Compiler	
Contract Name	Spiral
Optimization	Yes with 200 runs
LicenseType	UNLICENSED
Language	Solidity
Codebase	TBD

MainNet Contract was Not Assessed

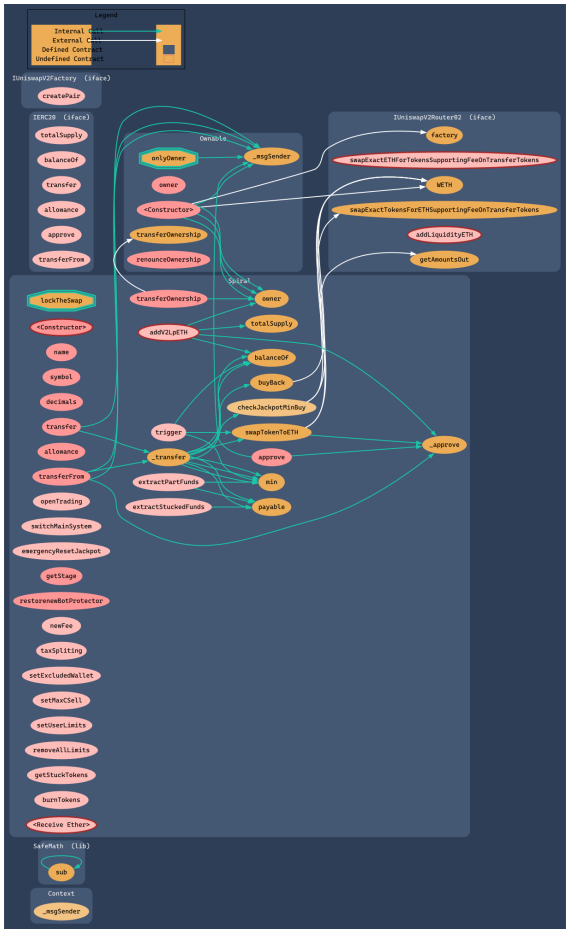
TestNet Contract Was Not Assessed

Solidity Code Provided

SolID	File Sha-1	FileName
Spiral	undefined	SPIRALCA.sol

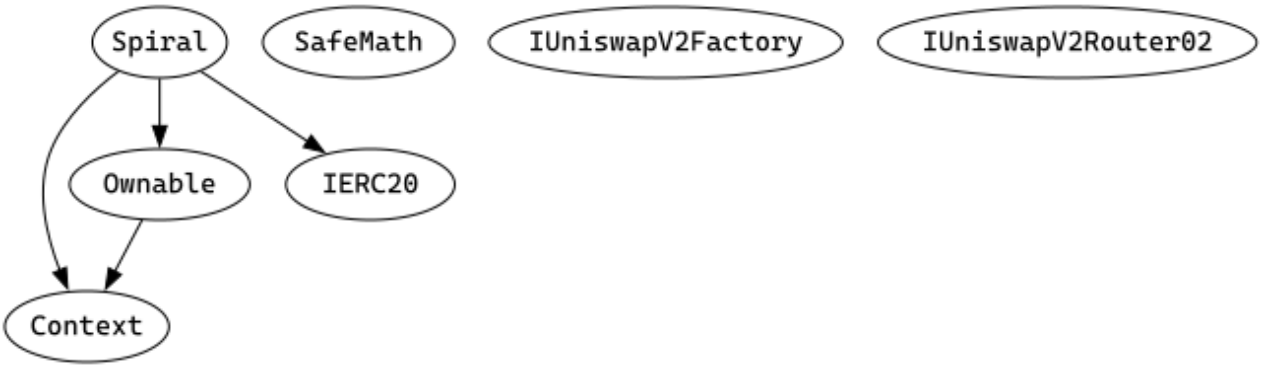
Call Graph

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 | Spiral.

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.

SPIRAL-02 | Function Visibility Optimization.

Category	Severity	Location	Status
Security	High	SPIRALCA.sol: L:478, C:9 and L:486, C:9	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

Owner should implement slippage protection in swapTokenToETH() and buyBack() functions

References:

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

SPIRAL-03 | Lack of Input Validation.

Category	Severity	Location	Status
Volatile Code	Medium	SPIRALCA.sol: L:495 (newFee), L:503 (setExcludedWallet), L:469 (restorenewBotProtector), L:513 (setMaxCSell), L:517 (setUserLimits)	Detected

Description

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

The given input is missing the check for the Add require statements for zero address checks and value range validations in all onlyOwner functions.

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. Add require statements for zero address checks and value range validations in all onlyOwner functions.

Mitigation

Owner must be trusted not to set malicious parameters

References:

Zero Address check. The danger!!!

SPIRAL-04 | Centralized Risk In addLiquidity.

Category	Severity	Location	Status
Centralization	High	SPIRALCA.sol: L:555-562 (addV2LpETH function)	Detected

Description

uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this), tokenAmount, 0, 0, owner(), block.timestamp);

The addLiquidity function calls the uniswapV2Router.addLiquidityETH function with the to address specified as owner() for acquiring the generated LP tokens from the SPIRAL-WBNB pool.

As a result, over time the _owner address will accumulate a significant portion of LP tokens.If the _owner is an EOA (Externally Owned Account), mishandling of its private key can have devastating consequences to the project as a whole.

Recommendation

We advise the to address of the uniswapV2Router.addLiquidityETH function call to be replaced by the contract itself, i.e. address(this) , and to restrict the management of the LP tokens within the scope of the contract's business logic. This will also protect the LP tokens from being stolen if the _owner account is compromised. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract based accounts with enhanced security practices, f.e. Multisignature wallets.

1. Indicatively, here are some feasible solutions that would also mitigate the potential risk:

2. Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;

3. Assignment of privileged roles to multi-signature wallets to prevent single point of failure due to the private key;

Introduction of a DAO / governance / voting module to increase transparency and user involvement

Mitigation

References:

Centralization Risk in Crypto: How Decentralized Is Crypto?

SPIRAL-05 | Missing Event Emission.

Category	Severity	Location	Status
Volatile Code	Medium	SPIRALCA.sol: L:495 (newFee), L:503 (setExcludedWallet), L:513 (setMaxCSell), L:517 (setUserLimits), L:522 (removeAllLimits), L:499 (taxSpliting), L:234 (switchMainSystem), L:469 (restorenewBotProtector)	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

Monitor owner wallet transactions directly on-chain

References:

Understanding Events in Smart Contracts

SPIRAL-06 | Conformance with Solidity Naming Conventions.

Category	Severity	Location	Status
Coding Style	Low	SPIRALCA.sol: L:469 (restorenewBotProtector), L:142 (newBotProtector), L:469 (newnewBotProtector parameter)	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.

restorenewBotProtector (L:469)
newBotProtector (L:142)
newnewBotProtector parameter (L:469)

Recommendation

Follow the Solidity naming convention.

Mitigation

Cosmetic issue, does not affect functionality

References:

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

Writing Clean Code for Solidity: Best Practices for Solidity Development

SPIRAL-08 | Dead Code Elimination.

Category	Severity	Location	Status
Code Quality	Low	SPIRALCA.sol: L:108 (holderId mapping), L:109 (holders array)	Detected

Description

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

```
holderId mapping (L:108)
holders array (L:109)
```

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

Minor gas impact on deployment only

References:

Cheatsheetl

Writing Clean Code for Solidity: Best Practices for Solidity Development

SPIRAL-09 | Third Party Dependencies.

Category	Severity	Location	Status
Volatile Code	Medium	SPIRALCA.sol: L:169 (Router initialization), L:478 (swapTokenToETH), L:486 (buyBack), L:555 (addLiquidityETH)	Detected

Description

The contract is serving as the underlying entity to interact with third party Uniswap V2 Router (0x4752ba5DBc23f44D87826276BF6Fd6b1C372aD24), Uniswap V2 Factory protocols. The scope of the audit treats 3rd party entities as black boxes and assume their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

Recommendation

We understand that the business logic of Spiral requires interaction with Uniswap V2 Router (0x4752ba5DBc23f44D87826276BF6Fd6b1C372aD24), Uniswap V2 Factory , etc. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

Mitigation

Monitor Uniswap V2 protocol and have contingency plans

Add router update capability and monitoring system

References:

Writing Clean Code for Solidity: Best Practices for Solidity Development

SPIRAL-10 | Initial Token Distribution.

Category	Severity	Location	Status
Centralization / Privilege	High	SPIRALCA.sol: L:177 (constructor)	Detected

Description

All of the Spiral tokens are sent to the contract deployer when deploying the contract. This could be a centralization risk as the deployer can distribute tokens without obtaining the consensus of the community.

Recommendation

We recommend the team to be transparent regarding the initial token distribution process, and the team shall make enough efforts to restrict the access of the private key.

Mitigation

Use multi-sig wallet and implement transparent distribution

Implement vesting and multi-sig control

References:

Writing Clean Code for Solidity: Best Practices for Solidity Development

SPIRAL-14 | Unnecessary Use Of SafeMath.

Category	Severity	Location	Status
Logical Issue	Low	SPIRALCA.sol: L:14-23 (SafeMath library), L:105 (using declaration), L:217 (usage)	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 Spiral contract.

Recommendation

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

Mitigation

Remove SafeMath import and usage

Refactor to use native arithmetic operators

References:

Writing Clean Code for Solidity: Best Practices for Solidity Development

SPIRAL-16 | Taxes can be up to 100%.

Category	Severity	Location	Status
Logical Issue	Critical	SPIRALCA.sol: L:495-497 (newFee function)	Detected

Description

The current definition of taxes can be set up to 100% for specific wallets, we suggest to modify the function not to be dynamic but to be a static resolution.

Add tax limit validation immediately

due to the logic written in here may results in a honeypot.

Recommendation

We advise the team to review the following logic..

Mitigation

Owner must commit to reasonable tax policy publicly

Add tax limit validation immediately

References:

Writing Clean Code for Solidity: Best Practices for Solidity Development

SPIRAL-17 | Highly Permissive Role Access.

Category	Severity	Location	Status
Logical Issue	High	SPIRALCA.sol: L:230-232 (openTrading), L:245 (jackpot check), L:261 (durationStage), L:301 (lastWinnerTime)	Detected

Description

The following is an example scenario of how the owner may manipulate the funds of accounts which do not belong to it.
During initialization, a set of "excluded" (via excludeFromReward(..)) accounts is created, whose balances - for the sake of argument - are small relative to the total supply, but are big enough in absolute values to consider those accounts rich.
Let = total reflection balance of the “excluded” accounts.
Let = total tax collected on all the transfers that were needed to set up the “excluded“ accounts.
Let
0 = ( ) −
Let = total token balance of the “excluded” accounts.
Let
0 = ( ) −
Then, an additional “excluded” account is introduced, whose reflection balance at the moment of exclusion was . After that, multiple transfers occur, with the total tax accrued equal to .
According to the formulas in the contract, the reflection-to-token exchange rate at this point is:
If now account gets "included" (via includeInReward(..)), the exchange rate becomes:
The ratio between the new and the old rate:

Introducing some convenience definitions:
It is possible to rewrite the expression as:
In other words, the holders of plain token i.e. "excluded" wallets, become % richer (in reflections), and the rest "not excluded" wallets become poorer by the respective percentage. Becoming richer for excluded wallets will not be immediately observable: for example,

the ERC20 balance of such an account will not increase as a number; nevertheless, it can be accounted via the contract interface, that those wallets actually became richer; eventually, the ecosystem would recognize who has become relatively richer/poorer.
It is possible to make the gain for the "excluded" wallets immediately obvious by "including" such accounts - that would lead to exchanging their tokens to reflections with the new rate; as a result, their ERC20 balances would reflect the gain literally.
Example: if is 8% of , and is 51.1% of , then the rate
0

0
increases by 10%.

Recommendation

Remove the functionality or provide documentation with its description.

Mitigation

Document timestamp manipulation risk in user-facing documentation

Add block.number based timing or external oracle

References:

Writing Clean Code for Solidity: Best Practices for Solidity Development

SPIRAL-18 | Stop Transactions by using Enable Trade.

Category	Severity	Location	Status
Logical Issue	High	SPIRALCA.sol: L:228-233 (openTrading), L:243 (launch check), L:503 (setExcludedWallet)	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

Publicly commit to fair launch without pre-trade exclusions

Remove dynamic exclusion or add immutability

References:

Writing Clean Code for Solidity: Best Practices for Solidity Development

SPIRAL-19 | Centralization Privileges of SPIRAL.

Category	Severity	Location	Status
Security	High	SPIRALCA.sol: L:300 (jackpot payout), L:526 (extractStuckedFunds), L:531 (extractPartFunds)	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
openTrading	none	external
switchMainSystem	none	external
emergencyResetJackpot	none	external
restorenewBotProtector	address	public
transferOwnership	address	public
newFee	uint256, uint256	external
taxSpliting	none	external
setExcludedWallet	address, bool	external
trigger	uint256	external
setMaxCSell	uint256	external

Function Name	Parameters	Visibility
setUserLimits	uint256, uint256	external
removeAllLimits	none	external
burnTokens	uint256	external
addV2LpETH	none	external payable

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 deployr or owner secure the credentials that has owner priviledge to ensure the security of the project.

Mitigation

Current gas limits provide some protection but not best practice

References:

Guide to Ownership and Access Control in Solidity

Writing Clean Code for Solidity: Best Practices for Solidity Development

SPIRAL-20 | Centralization Risk in Launch Mechanism.

Category	Severity	Location	Status
Centralization	Critical	SPIRALCA.sol: launch(), enableTransfer(), disableTransfer() functions	Detected

Description

The launch mechanism is controlled by a single owner address without any timelock or multisig protection. The owner has unilateral power to enable/disable transfers and control the launch state..

Recommendation

Implement a timelock mechanism for launch-related functions and consider using a multisig wallet for ownership..

Mitigation

References:

Writing Clean Code for Solidity: Best Practices for Solidity Development

SPIRAL-21 | Missing Access Control Recovery.

Category	Severity	Location	Status
Access Control	High	SPIRALCA.sol: Ownable implementation	Detected

Description

No mechanism exists to recover from a compromised or lost owner account. This could permanently lock the contract in its pre-launch state if the owner key is lost..

Recommendation

Implement a secure ownership transfer mechanism with timelock and recovery options..

Mitigation

References:

Writing Clean Code for Solidity: Best Practices for Solidity Development

SPIRAL-24 | Missing Input Validation.

Category	Severity	Location	Status
Input Validation	Low	SPIRALCA.sol: enableTransfer(), disableTransfer() functions	Detected

Description

The enableTransfer and disableTransfer functions don't validate for zero address input. This could lead to unnecessary gas consumption and confusing state..

Recommendation

Add zero address validation in transfer right management functions..

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	2	1	0
High	4	6	0
Medium	4	3	0
Low	5	3	0
Informational	1	0	0
Total	16	13	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 | Spiral.

Social Media	URL	Result
Website	https://spiraleth.xyz	Pass
Telegram	https://t.me/SpiralOnEth	Pass
Twitter	https://x.com/spiralEthX	Pass
Facebook		N/A
Reddit	N/A	N/A
Instagram	N/A	N/A
CoinGecko		Fail
Github	N/A	N/A
CMC		Fail
Email		Contact
Other	https://docs.spiraleth.xyz	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: Complete social media presence with website, documentation, Telegram, and X.

Project Owner Notes: Active community engagement across platforms.

Assessment Results | Final Audit Score SPIRAL.

Review	Score
Security Score	48
Auditor Score	48

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.

AUDIT FAILED

Important Notes for SPIRAL

- Spiral Token (SPIRAL) Audit Report
- Contract Type: ERC20 with Jackpot Mechanism ("LastBuyerWins")
- Platform: Base Chain
- Compiler Version: 0.8.24
- Audit Date: January 8, 2026
- Contract Overview:
- Spiral is an experimental ERC20 token with a jackpot system where the last buyer wins accumulated ETH rewards. Features include buyback mechanisms, adjustable taxes, and Uniswap V2 integration on Base Chain.
- Security Classification:
- Audit Score: 48/100 (FAIL - Below 80 threshold)
- Confidence Level: Very High Risk
- Centralization Risk: Critical (Unlimited owner control)
- Immediate Actions Required:
- 1. DO NOT DEPLOY - Critical security risks present
- 2. Add MAX_TAX constant (10%) and enforce in newFee() function
- 3. Install OpenZeppelin ReentrancyGuard on all ETH transfer functions
- 4. Implement 0.5-1% slippage protection on Uniswap operations
- 5. Lock LP tokens in verified time-lock contract (6-12 months minimum)
- 6. Implement token vesting for team allocation
- 7. Add automated launch system with public announcement
- 8. Replace block.timestamp with block.number for jackpot timing
- 9. Add input validation and events to all owner functions
- Recommendations:

- Timeline Estimate: 4-6 weeks to implement fixes and conduct professional audit before safe deployment.
- CFGNinja strongly recommends implementing ALL critical fixes before any public deployment or marketing activities.

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.