

CHENINJA AUDITS



Security Assessment

**Federal Reserve 3.0
Token**

August 22, 2022



Table of Contents

1 Audit Summary

2 Project Overview

2.1 Token Summary

2.2 Risk Analysis Summary

2.3 Main Contract Assessed

3 Smart Contract Risk Checks

3.1 Mint Check

3.2 Fees Check

3.3 Blacklist Check

3.4 MaxTx Check

3.5 Pause Trade Check

4 Contract Ownership

5 Liquidity Ownership

6 KYC Check

7 Smart Contract Vulnerability Checks

7.1 Smart Contract Vulnerability Details

7.2 Smart Contract Inheritance Details

7.3 Smart Contract Privileged Functions

8 Assessment Results and Notes(Important)

9 Social Media Check(Informational)

10 Technical Findings Summary

11 Disclaimer



Audit Summary

This report has been prepared for Federal Reserve 3.0 Token on the Binance Smart Chain network. CFGNINJA provides both client-centered and user-centered examination of the smart contracts and their current status when applicable. This report represents the security assessment made to find issues and vulnerabilities on the source code along with the current liquidity and token holder statistics of the protocol.

A comprehensive examination has been performed, utilizing Cross Referencing, Static Analysis, In-House Security Tools, and line-by-line Manual Review.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Inspecting liquidity and holders statistics to inform the current status to both users and client when applicable.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Verifying contract functions that allow trusted and/or untrusted actors to mint, lock, pause, and transfer assets.



Project Overview

Token Summary

| Parameter | Result |
|---------------|---|
| Address | 0x83df572468b836f70d166c73a631b708cc18b0f9 |
| Name | Federal Reserve 3.0 |
| Token Tracker | Federal Reserve 3.0 (\$FED3.0) |
| Decimals | 18 |
| Supply | 100,000,000,000 |
| Platform | Binance Smart Chain |
| compiler | v0.8.16+commit.07a7930e |
| Contract Name | FederalReserve |
| Optimization | Yes with 200 runs |
| LicenseType | MIT |
| Language | Solidity |
| Codebase | https://bscscan.com/address/0x83df572468b836f70d166c73a631b708cc18b0f9#code |
| Payment Tx | 0x950471ff1f9dc6022bde3852b8dfec1a22dad84fe4b67383660c97dbd41e2209 |



Project Overview

Risk Analysis Summary

| Parameter | Result |
|------------------|--------|
| Buy Tax | 10% |
| Sale Tax | 10% |
| Is honeypot? | Clean |
| Can edit tax? | Yes |
| Is anti whale? | No |
| Is blacklisted? | Yes |
| Is whitelisted? | No |
| Holders | Clean |
| Security Score | 95/100 |
| Auditor Score | 96/100 |
| Confidence Level | Pass |

The following quick summary has been added to the project overview, however there are more details about the audit and their results please read every details.



Main Contract Assessed Contract Name

| Name | Contract | Live |
|---------------------|--|------|
| Federal Reserve 3.0 | 0x83df572468b836f70d166c73a631b708cc18b0f9 | Yes |

TestNet Contract Assessed Contract Name

| Name | Contract | Live |
|---------------------|--|------|
| Federal Reserve 3.0 | 0x5926CE5DBc4cC9A712Bb78e8F2752071675a2E42 | Yes |

Solidity Code Provided

| SolID | File Sha-1 | FileName |
|----------------|--|--------------------|
| FederalReserve | b3aacbbfe7998b3d8c0ebf84f1f62d2a8664a5cf | FederalReserve.sol |



Mint Check

The Project Owners of Federal Reserve 3.0 does not have a mint function in the contract, owner cannot mint tokens after initial deploy

..

The Project has a Total Supply of 100,000,000,000 and cannot mint any more than the Max Supply.

.

Mint Notes:

Auditor Notes: A Mint Function was not found during the code review

Project Owner Notes:



Owner can't mint new coins



Fees Check

The Project Owners of Federal Reserve 3.0 does not have the ability to set fees higher than 25% .

Team May have fees defined, however they dont have the ability to set those fees higher than 25%.

Tax Fee Notes:

Auditor Notes: Contract currently have 10% tax and can be increased up to 15%

Project Owner Notes: .



Fees can be changed up to a maximum of 25%



Blacklist Check

The Project Onwers of Federal Reserve 3.0 has the ability to Blacklist holders from transferring their tokens.

We Recommend the team to be careful with a blacklist function as this can basically prevent ah holder from buying/selling/transferring their assets. Malicious or compromised owners can trap contracts relying on tokens with a blacklist.

Blacklist Notes:

Auditor Notes: Contract have a blacklist function presented.

Project Owner Notes: .



MaxTx Check

The Project Owners of Federal Reserve 3.0 can set max tx amount.

The ability to set MaxTx can be used as bad actor, this can limit the ability of investors to sale their tokens at any given time if is set too low..

We recommend the project to set MaxTx to Total Supply or simiar to avoid swap or transfer from failures

MaxTX Notes:

Auditor Notes: Project Owner can set maxTx and maxWallet, this is currently set to total supply

Project Owner Notes: Project Owner state. cannot reduce it lower than 0.1%



Pause Trade Check

The Project Owners of Federal Reserve 3.0 don't have the ability to stop or pause trading.

The Team has done a great job to avoid stop trading, and investors has the ability to trade at any given time without any problems

Pause Trade Notes:

Auditor Notes: Project Owner can only start the trade and don't have the ability to stop the trade after that. function `startTrading(bool_bool)` external authorized.

Project Owner Notes: The Start Trade will be executed right after launch and no longer have the ability to stop the trade.



Owner can't pause trading



Contract Ownership

The contract ownership of Federal Reserve 3.0 is not currently renounced. The ownership of the contract grants special powers to the protocol creators, making them the sole addresses that can call sensible ownable functions that may alter the state of the protocol.

The current owner is the address
0xc4b64e8785a670b42d72a3d4b58fae8453c12002
which can be viewed from:
[HERE](#)

The owner wallet has the power to call the functions displayed on the privileged functions chart below, if the owner wallet is compromised this privileges could be exploited.

We recommend the team to renounce ownership at the right timing if possible, or gradually migrate to a timelock with governing functionalities in respect of transparency and safety considerations.

We recommend the team to use a Multisignature Wallet if contract is not going to be renounced, this will give the ability to the team to have more control over the contract.



Liquidity Ownership

The token does not have liquidity at the moment of the audit, block 20637265

If liquidity is unlocked, then the token developers can do what is infamously known as 'rugpull'. Once investors start buying token from the exchange, the liquidity pool will accumulate more and more coins of established value (e.g., ETH or BNB or Tether). This is because investors are basically sending these tokens of value to the exchange, to get the new token. Developers can withdraw this liquidity from the exchange, cash in all the value and run off with it. Liquidity is locked by renouncing the ownership of liquidity pool (LP) tokens for a fixed time period, by sending them to a time-lock smart contract. Without ownership of LP tokens, developers cannot get liquidity pool funds back. This provides confidence to the investors that the token developers will not run away with the liquidity money. It is now a standard practice that all token developers follow, and this is what really differentiates a scam coin from a real one.

[Read More](#)



KYC Information

The Project Owners of Federal Reserve 3.0 has provided KYC Documentation.

KYC Certificated can be found on the Following:
KYC Data

KYC Information Notes:

Auditor Notes: Asked project owner about KYC, Project owner is waiting for contract deployment to pass IDOPresales

Project Owner Notes:



Smart Contract Vulnerability Checks

| ID | Severity | Name | File | location |
|---------|----------|---|--------------------|--------------|
| SWC-100 | Pass | Function Default Visibility | FederalReserve.sol | L: 0 C: 0 |
| SWC-101 | Pass | Integer Overflow and Underflow. | FederalReserve.sol | L: 0 C: 0 |
| SWC-102 | Pass | Outdated Compiler Version file. | FederalReserve.sol | L: 0 C: 0 |
| SWC-103 | Pass | A floating pragma is set. | FederalReserve.sol | L: 2 C: 0 |
| SWC-104 | Pass | Unchecked Call Return Value. | FederalReserve.sol | L: 0 C: 0 |
| SWC-105 | Pass | Unprotected Ether Withdrawal. | FederalReserve.sol | L: 0 C: 0 |
| SWC-106 | Pass | Unprotected SELFDESTRUCT Instruction | FederalReserve.sol | L: 0 C: 0 |
| SWC-107 | Pass | Read of persistent state following external call. | FederalReserve.sol | L: 0 C: 0 |
| SWC-108 | Pass | State variable visibility is not set.. | FederalReserve.sol | L: 290 C: 12 |
| SWC-109 | Pass | Uninitialized Storage Pointer. | FederalReserve.sol | L: 0 C: 0 |
| SWC-110 | Pass | Assert Violation. | FederalReserve.sol | L: 0 C: 0 |
| SWC-111 | Pass | Use of Deprecated Solidity Functions. | FederalReserve.sol | L: 0 C: 0 |
| SWC-112 | Pass | Delegate Call to Untrusted Callee. | FederalReserve.sol | L: 0 C: 0 |



| ID | Severity | Name | File | location |
|---------|----------|--|--------------------|--|
| SWC-113 | Pass | Multiple calls are executed in the same transaction. | FederalReserve.sol | L: 0 C: 0 |
| SWC-114 | Pass | Transaction Order Dependence. | FederalReserve.sol | L: 0 C: 0 |
| SWC-115 | Pass | Authorization through tx.origin. | FederalReserve.sol | L: 0 C: 0 |
| SWC-116 | Pass | A control flow decision is made based on The block.timestamp environment variable. | FederalReserve.sol | L: 0 C: 0 |
| SWC-117 | Pass | Signature Malleability. | FederalReserve.sol | L: 0 C: 0 |
| SWC-118 | Pass | Incorrect Constructor Name. | FederalReserve.sol | L: 0 C: 0 |
| SWC-119 | Pass | Shadowing State Variables. | FederalReserve.sol | L: 0 C: 0 |
| SWC-120 | Low | Potential use of block.number as source of randomness. | FederalReserve.sol | L: 421 C: 12, L: 496 C: 22, L: 497 C: 28 |
| SWC-121 | Pass | Missing Protection against Signature Replay Attacks. | FederalReserve.sol | L: 0 C: 0 |
| SWC-122 | Pass | Lack of Proper Signature Verification. | FederalReserve.sol | L: 0 C: 0 |
| SWC-123 | Pass | Requirement Violation. | FederalReserve.sol | L: 0 C: 0 |
| SWC-124 | Pass | Write to Arbitrary Storage Location. | FederalReserve.sol | L: 0 C: 0 |
| SWC-125 | Pass | Incorrect Inheritance Order. | FederalReserve.sol | L: 0 C: 0 |
| SWC-126 | Pass | Insufficient Gas Griefing. | FederalReserve.sol | L: 0 C: 0 |



| ID | Severity | Name | File | location |
|---------|----------|--|--------------------|-----------|
| SWC-127 | Pass | Arbitrary Jump with Function Type Variable. | FederalReserve.sol | L: 0 C: 0 |
| SWC-128 | Pass | DoS With Block Gas Limit. | FederalReserve.sol | L: 0 C: 0 |
| SWC-129 | Pass | Typographical Error. | FederalReserve.sol | L: 0 C: 0 |
| SWC-130 | Pass | Right-To-Left-Override control character (U+202E). | FederalReserve.sol | L: 0 C: 0 |
| SWC-131 | Pass | Presence of unused variables. | FederalReserve.sol | L: 0 C: 0 |
| SWC-132 | Pass | Unexpected Ether balance. | FederalReserve.sol | L: 0 C: 0 |
| SWC-133 | Pass | Hash Collisions with Multiple Variable Length Arguments. | FederalReserve.sol | L: 0 C: 0 |
| SWC-134 | Pass | Message call with hardcoded gas amount. | FederalReserve.sol | L: 0 C: 0 |
| SWC-135 | Pass | Code With No Effects (Irrelevant/Dead Code). | FederalReserve.sol | L: 0 C: 0 |
| SWC-136 | Pass | Unencrypted Private Data On-Chain. | FederalReserve.sol | L: 0 C: 0 |

We scan the contract for additional security issues using MYTHX and industry standard security scanning tool



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?)

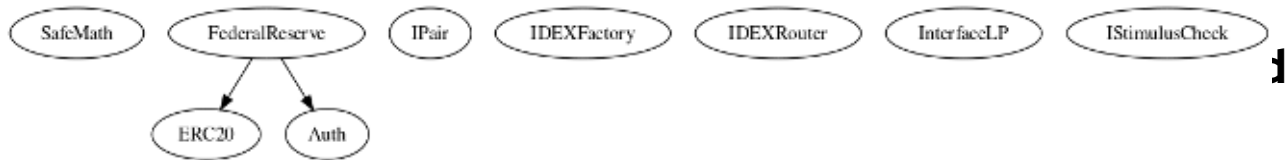
When can BLOCKHASH be safely used for a random number? When would it be unsafe?

The Run smart contract.



Call Graph and Inheritance

The contract for Federal Reserve 3.0 has the following call graph structure



.



Privileged Functions (onlyOwner)

| Function Name | Parameters | Visibility |
|-----------------------|------------|------------|
| authorize | none | public |
| unauthorize | none | public |
| transferOwnership | none | public |
| clearStuckBalance | none | public |
| tradingStatus | none | external |
| setTxLimit | none | external |
| setMaxWallet | none | external |
| manage_blacklist | none | external |
| setBL | none | external |
| updateF | none | public |
| updateMarketingWallet | none | external |



| Function Name | Parameters | Visibility |
|---------------------------------|------------|------------|
| setIsFeeExempt | none | external |
| setIsMaxWalletExempt | none | external |
| setIsTxLimitExempt | none | external |
| setBuyFees | none | external |
| setSellFees | none | external |
| setTransferFee | none | external |
| setFeeReceivers | none | external |
| setSwapBackSettings | none | external |
| setAutomatedMarket MakerPair | none | external |
| setStimulusCA | none | external |



| Function Name | Parameters | Visibility |
|------------------------------|------------|------------|
| updateStimulusStatus | none | external |
| updateStimulusCooldown | none | external |
| updateStimulusCheckThreshold | none | external |
| rescueERC20 | none | external |



Assessment Results

- Project Team seems to be doing the due diligence to ensure a successful launch.
- Contract has been tested and we did not found any potential errors or issues with the contract on our latest test.
- <https://testnet.bscscan.com/address/0x2AAFe9e0B236895c631f63628b552e9d15168785>
- Project Owner has the ability to authorize more than one wallet to execute the traditional onlyOwner functions.

Audit Passed



Social Media Checks

| Social Media | URL | Result |
|--------------|---|--------|
| Twitter | https://twitter.com/the_degen_fed | Pass |
| Instagram | https://www.tiktok.com/@the_degen_fed | Pass |
| Website | http://www.fedreserve.money/ | Pass |
| Telegram | https://t.me/fedreserveportal | Pass |

We recommend to have 3 or more social media sources including a completed working websites.

Social Media Information Notes:






Auditor Notes: undefined

Project Owner Notes: No other social media








Technical Findings Summary

Classification of Risk



| Severity | Description |
|---|---|
|  Critical | risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks. |
|  Major | risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project. |
|  Medium | risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform |
|  Minor | risks can be any of the above but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions. |
|  Informational | errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code. |

Findings

| Severity | Found | Pending | Resolved |
|---|-------|---------|----------|
|  Critical | 0 | 0 | 0 |
|  Major | 1 | 0 | 0 |
|  Medium | 1 | 0 | 0 |
|  Minor | 1 | 0 | 0 |
|  Informational | 1 | 1 | 0 |
| Total | 1 | 1 | 0 |



\$FED3.0-01 | Potential Sandwich Attacks.

| Category | Severity | Location | Status |
|----------|--|-------------------------------|---|
| Security |  Medium | FederalReserve.sol: 806,13 |  Pending |

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()
- addLiquidityETH()

Remediation



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

References:

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



\$FED3.0-02 | Function Visibility Optimization.

| Category | Severity | Location | Status |
|------------------|---|-------------------------|---|
| Gas Optimization |  Informational | FederalReserve.sol: 0,0 |  Pending |

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 |
|---------------|------------|------------|
| name | | public |
| symbol | | public |
| decimals | | public |
| setTaxes | | public |
| setSellTaxes | | public |

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

Remediation



We advise that the functions' 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.

References:

external vs public best practices.



\$FED3.0-04 | Centralized Risk In addLiquidity.

| Category | Severity | Location | Status |
|--------------|---|-------------------------------|---|
| Coding Style |  Major | FederalReserve.sol: 720,12 |  Pending |

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 \$FED3.0-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.

Remediation

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



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

CFGNINJA has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocacy for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and CFGNINJA is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will CFGNINJA or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

The assessment services provided by CFGNINJA is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

