Coinsult

Advanced Manual
Smart Contract Audit

November 24, 2025

🐦 CoinsultAudits

✈ t.me/coinsult_tg

🌐 coinsult.net

Audit requested by

ViFoxCoin

0xec69f1351b66902cd5e79f0924a5ad049f682540

Global Overview

Manual Code Review

In this audit report we will highlight the following issues:

Vulnerability Level	Total	Pending	Acknowledged	Resolved
● Informational	0	0	0	0
● Low-Risk	3	0	3	0
● Medium-Risk	2	0	2	0
● High-Risk	0	0	0	0

Centralization Risks

Coinsult checked the following privileges:

Contract Privilege	Description
Owner needs to enable trading?	● Owner needs to manually enable trading
Owner can mint?	● Owner can mint new tokens
Owner can blacklist?	● Owner cannot blacklist addresses
Owner can set fees?	● Owner can set the sell fee to 0%
Owner can exclude from fees?	● Owner cannot exclude from fees
Can be honeypotted?	● Contract can become honeypot
Owner can set Max TX amount?	● Owner cannot set max transaction amount

More owner priviliges are listed later in the report.

Table of Contents

Audit Summary

Project Name	ViFoxCoin
Website	https://ViFoxCoin.com
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	0xec69f1351b66902cd5e79f0924a5ad049f682540
Audit Method	Static Analysis, Manual Review
Date of Audit	24 November 2025

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Audit Scope

Coinsult was comissioned by ViFoxCoin to perform an audit based on the following code:

https://bscscan.com/token/0xec69f1351b66902cd5e79f0924a5ad049f682540#code

Note that we only audited the code available to us on this URL at the time of the audit. If the URL is not from any block explorer (main net), it may be subject to change. Always check the contract address on this audit report and compare it to the token you are doing research for.

Audit Method

Coinsult's manual smart contract audit is an extensive methodical examination and analysis of the smart contract's code that is used to interact with the blockchain. This process is conducted to discover errors, issues and security vulnerabilities in the code in order to suggest improvements and ways to fix them.

Automated Vulnerability Check

Coinsult uses software that checks for common vulnerability issues within smart contracts. We use automated tools that scan the contract for security vulnerabilities such as integer-overflow, integer-underflow, out-of-gas-situations, unchecked transfers, etc.

Manual Code Review

Coinsult's manual code review involves a human looking at source code, line by line, to find vulnerabilities. Manual code review helps to clarify the context of coding decisions. Automated tools are faster but they cannot take the developer's intentions and general business logic into consideration.

Used tools

- Slither: Solidity static analysis framework
- Remix: IDE Developer Tool
- CWE: Common Weakness Enumeration
- SWC: Smart Contract Weakness Classification and Test Cases
- DEX: Testnet Blockchains

Risk Classification

Coinsult uses certain vulnerability levels, these indicate how bad a certain issue is. The higher the risk, the more strictly it is recommended to correct the error before using the contract.

Vulnerability Level	Description
🔵 Informational	Does not compromise the functionality of the contract in any way
🟢 Low-Risk	Won't cause any problems, but can be adjusted for improvement
🟡 Medium-Risk	Will likely cause problems and it is recommended to adjust
🔴 High-Risk	Will definitely cause problems, this needs to be adjusted

Coinsult has four statuses that are used for each risk level. Below we explain them briefly.

Risk Status	Description
Total	Total amount of issues within this category
Pending	Risks that have yet to be addressed by the team
Acknowledged	The team is aware of the risks but does not resolve them
Resolved	The team has resolved and remedied the risk

SWC Attack Analysis

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	Description	Status
SWC-100	Function Default Visibility	Passed
SWC-101	Integer Overflow and Underflow	Passed
SWC-102	Outdated Compiler Version	Passed
SWC-103	Floating Pragma	Passed
SWC-104	Unchecked Call Return Value	Passed
SWC-105	Unprotected Ether Withdrawal	Passed
SWC-106	Unprotected SELFDESTRUCT Instruction	Passed
SWC-107	Reentrancy	Passed
SWC-108	State Variable Default Visibility	Passed
SWC-109	Uninitialized Storage Pointer	Passed
SWC-110	Assert Violation	Passed
SWC-111	Use of Deprecated Solidity Functions	Passed
SWC-112	Delegatecall to Untrusted Callee	Passed
SWC-113	DoS with Failed Call	Passed
SWC-114	Transaction Order Dependence	Passed
SWC-115	Authorization through tx.origin	Passed

SWC-116	Block values as a proxy for time	Passed
SWC-117	Signature Malleability	Passed
SWC-118	Incorrect Constructor Name	Passed
SWC-119	Shadowing State Variables	Passed
SWC-120	Weak Sources of Randomness from Chain Attributes	Passed
SWC-121	Missing Protection against Signature Replay Attacks	Passed
SWC-122	Lack of Proper Signature Verification	Passed
SWC-123	Requirement Violation	Passed
SWC-124	Write to Arbitrary Storage Location	Passed
SWC-125	Incorrect Inheritance Order	Passed
SWC-126	Insufficient Gas Griefing	Passed
SWC-127	Arbitrary Jump with Function Type Variable	Passed
SWC-128	DoS With Block Gas Limit	Passed
SWC-129	Typographical Error	Passed
SWC-130	Right-To-Left-Override control character (U+202E)	Passed
SWC-131	Presence of unused variables	Passed
SWC-132	Unexpected Ether balance	Passed
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Passed
SWC-134	Message call with hardcoded gas amount	Passed
SWC-135	Code With No Effects	Passed
SWC-136	Unencrypted Private Data On-Chain	Passed

Error Code	Description
SLT: 078	Conformance to numeric notation best practices

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Too many digits

Literals with many digits are difficult to read and review.

```
uint256 public maxSupply = 21000000 * 10 ** DECIMALS;
uint256[5] memory thresholds = [
            11000000 * 10 ** DECIMALS,
            9000000 * 10 ** DECIMALS,
            7000000 * 10 ** DECIMALS,
            5000000 * 10 ** DECIMALS,
            3000000 * 10 ** DECIMALS
      ];
```

Recommendation

Use: Ether suffix, Time suffix, or The scientific notation

Exploit scenario

```
contract MyContract{
    uint 1_ether = 10000000000000000000;
}
```

While 1_ether looks like 1 ether, it is 10 ether. As a result, it's likely to be used incorrectly.

Error Code	Description
SLT: 054	Missing Events Arithmetic

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```solidity
function setDailyMintLimit(uint256 newLimit) external onlyRole(ADMIN_ROLE) {
    dailyMintLimit = newLimit;
}

function defineOperator(address operator) external onlyRole(ADMIN_ROLE) {
    grantRole(MINTER_ROLE, operator);
}

function changeOperator(address oldOperator, address newOperator) external onlyRole(ADMIN_ROLE) {
    revokeRole(MINTER_ROLE, oldOperator);
    grantRole(MINTER_ROLE, newOperator);
}

function removeOperator(address operator) external onlyRole(ADMIN_ROLE) {
    revokeRole(MINTER_ROLE, operator);
    emit OperatorRemoved(operator);
}
```

Recommendation

Emit an event for critical parameter changes.

Exploit scenario

```solidity
contract C {

  modifier onlyAdmin {
    if (msg.sender != owner) throw;
    _;
  }

  function updateOwner(address newOwner) onlyAdmin external {
    owner = newOwner;
  }
}
```

updateOwner() has no event, so it is difficult to track off-chain changes in the buy price.

Error Code	Description
CS: 016	Initial Supply

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Initial Supply

When the contract is deployed, the contract deployer receives all of the initially created assets. Since the deployer and/or contract owner can distribute tokens without consulting the community, this could be a problem.

Recommendation

Private keys belonging to the employer and/or contract owner should be stored properly. The initial asset allocation procedure should involve consultation with the community.

Error Code	Description
CSM-01	Vesting can permanently lock non-vested tokens

🟡 **Medium-Risk:** Should be fixed, could bring problems.

Vesting can permanently lock non-vested tokens

```
function vestedAmount(address account) public view returns (uint256) {
    Vesting memory vest = vestings[account];
    if (vest.totalAllocation == 0) return balanceOf(account);
    ...
}

function transferableAmount(address account) public view returns (uint256) {
    Vesting memory vest = vestings[account];
    if (vest.totalAllocation == 0) return balanceOf(account); // not vested

    uint256 vested = vestedAmount(account);
    uint256 alreadyReleased = vest.released;
    if (vested  bal ? bal : allowed;
}

function _update(address from, address to, uint256 amount)
    internal
    virtual
    override(ERC20Pausable)
{
    if (from != address(0) &&& to != address(0)) {
```

Recommendation

For accounts that have a Vesting struct, all transfers out of that account are globally limited by totalAllocation, regardless of where the tokens came from. So if address X initially vested 1000 tokens and 100 tokens are send to address x. These 100 tokens are now locked in account X and cannot be send anywhere else again. Seperate the vested amount from the total balance amount to solve this issue.

Error Code	Description
CSM-02	Inefficient usntake function

🟡 **Medium-Risk:** Should be fixed, could bring problems.

Inefficient usntake function

```
Stake[] storage stakes = userStakes[msg.sender];

for (uint256 i = 0; i  0; i++) {
    ...
}
...
for (uint256 i = stakes.length; i &gt; 0; i--) {
    if (stakes[i - 1].amount == 0) stakes.pop();
    else break;
}
```

Recommendation

If a user creates very many small stakes, the length of userStakes[msg.sender] can grow large, making unstake gas-heavy and potentially making it fail due to out-of-gas problem.

Maximum Fee Limit Check

Error Code	Description
CEN-01	Centralization: Operator Fee Manipulation

Coinsult tests if the owner of the smart contract can set the transfer, buy or sell fee to 25% or more. It is bad practice to set the fees to 25% or more, because owners can prevent healthy trading or even stop trading when the fees are set too high.

Type of fee	Description
Max transfer fee	0%
Max buy fee	0%
Max sell fee	0%

Contract Honeypot Check

Error Code	Description
CEN-02	Centralization: Operator Pausability

Coinsult tests if the owner of the smart contract has the ability to pause the contract. If this is the case, users can no longer interact with the smart contract; users can no longer trade the token.

Privilege Check	Description
Can owner pause the contract?	🔴 Owner can pause the smart contract

Function

```
function pause() external onlyRole(ADMIN_ROLE) {
    _pause();
}
function unpause() external onlyRole(ADMIN_ROLE) {
    _unpause();
}
```

Max Transaction Amount Check

Error Code	Description
CEN-03	Centralization: Operator Transaction Manipulation

Coinsult tests if the owner of the smart contract can set the maximum amount of a transaction. If the transaction exceeds this limit, the transaction will revert. Owners could prevent normal transactions to take place if they abuse this function.

Privilege Check	Description
Can owner set max tx amount?	🟢 Owner cannot set max transaction amount

Exclude From Fees Check

Error Code	Description
CEN-04	Centralization: Operator Exclusion

Coinsult tests if the owner of the smart contract can exclude addresses from paying tax fees. If the owner of the smart contract can exclude from fees, they could set high tax fees and exclude themselves from fees and benefit from 0% trading fees. However, some smart contracts require this function to exclude routers, dex, cex or other contracts / wallets from fees.

Privilege Check	Description
Can owner exclude from fees?	🟢 Owner cannot exclude from fees

Ability To Mint Check

Error Code	Description
CEN-05	Centralization: Operator Increase Supply

Coinsult tests if the owner of the smart contract can mint new tokens. If the contract contains a mint function, we refer to the token's total supply as non-fixed, allowing the token owner to "mint" more tokens whenever they want.

A mint function in the smart contract allows minting tokens at a later stage. A method to disable minting can also be added to stop the minting process irreversibly.

Minting tokens is done by sending a transaction that creates new tokens inside of the token smart contract. With the help of the smart contract function, an unlimited number of tokens can be created without spending additional energy or money.

Privilege Check	Description
Can owner mint?	🔴 Owner can mint new tokens

Function

```
function transferVFX(uint256 userId, uint256 vfxPoints, address userWallet)
external onlyRole(MINTER_ROLE) ...
```

Enable Trading

Error Code	Description
CEN-06	Centralization: Operator enable trading

Coinsult tests if the owner of the smart contract needs to manually enable trading before everyone can buy & sell. If the owner needs to manually enable trading, this poses a high centralization risk.

If the owner needs to manually enable trading, make sure to check if the project has a SAFU badge or a trusted KYC badge. Always DYOR when investing in a project that needs to manually enable trading.

Privilege Check	Description
Owner needs to enable trading?	🔴 Owner needs to manually enable trading

Ability To Blacklist Check

Error Code	Description
CEN-07	Centralization: Operator Dissalows Wallets

Coinsult tests if the owner of the smart contract can blacklist accounts from interacting with the smart contract. Blacklisting methods allow the contract owner to enter wallet addresses which are not allowed to interact with the smart contract.

This method can be abused by token owners to prevent certain / all holders from trading the token. However, blacklists might be good for tokens that want to rule out certain addresses from interacting with a smart contract.

Privilege Check	Description
Can owner blacklist?	🟢 Owner cannot blacklist addresses

Other Owner Privileges Check

Error Code	Description
CEN-100	Centralization: Operator Priviliges

Coinsult lists all important contract methods which the owner can interact with.

✅ No other important owner privileges to mention.

Coinsult

Notes

Notes by ViFoxCoin

Response to CSM-01:
The vesting mechanism is exclusively designed for initial token allocations to founding partners and participating companies. These addresses are predetermined and controlled, and no external token transfers will be made to them during the project lifecycle. Since these vested addresses are not intended to receive any additional tokens from third parties, the risk of accidentally locking non-vested tokens does not apply in our use case. We acknowledge this finding and have taken it into consideration for our operational procedures.

Response to CSM-02:
We have implemented a minimum staking threshold of 50 tokens at the UI level to prevent users from creating multiple small stakes. This measure significantly reduces the likelihood of gas-heavy unstake operations, as users interacting through our platform cannot create stakes below this limit. While this restriction is enforced on the frontend, we acknowledge that direct contract interactions could bypass this limitation. However, given that the majority of our users interact through the official website, this mitigation effectively addresses the practical risk.

Response to CS-016:

The initial token supply is distributed directly during deployment to 11 participating partner companies, each receiving their allocation to their designated wallets. Furthermore, these allocations are subject to vesting schedules, ensuring a controlled and gradual release of tokens over time. This approach eliminates the centralization risk of a single deployer holding all tokens and demonstrates our commitment to fair and transparent token distribution from the very beginning.

Response to CEN-02:
The contract's admin role has been transferred to a multi-signature wallet requiring 2 out of 3 signatures for any administrative action, including pausing the contract. This ensures that no single individual can unilaterally pause trading or execute other critical functions. The multi-sig setup provides an additional layer of security and decentralization, protecting users from potential misuse of admin privileges.

Notes by Coinsult
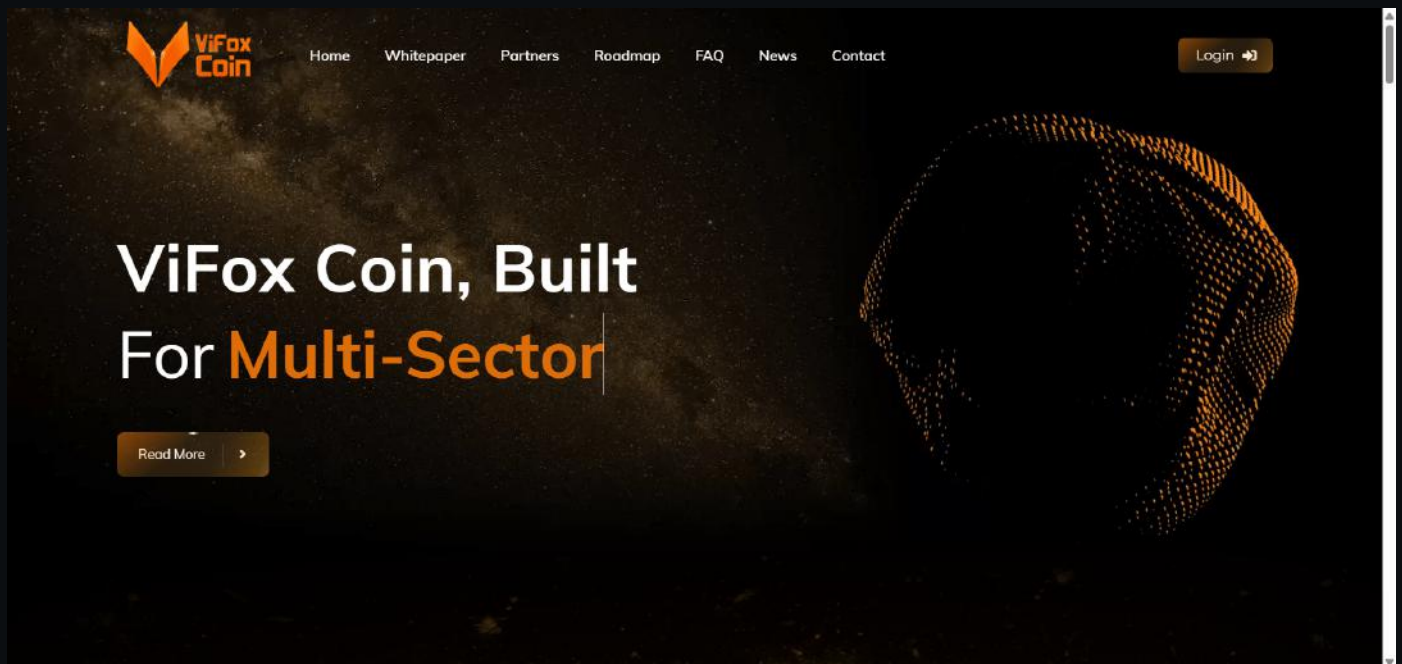
No notes provided by Coinsult

Contract Snapshot

This is how the constructor of the contract looked at the time of auditing the smart contract.

```
contract ViFoxCoin is ERC20Pausable, AccessControl, ReentrancyGuard {
uint256 public constant DECIMALS = 3;
uint256 public maxSupply = 21000000 * 10 ** DECIMALS;
uint256 public constant MIN_VFX_FOR_TRANSFER = 1 * 10 ** DECIMALS;
```

Website Review

Coinsult checks the website completely manually and looks for visual, technical and textual errors. We also look at the security, speed and accessibility of the website. In short, a complete check to see if the website meets the current standard of the web development industry.

Type of check	Description
Mobile friendly?	🟢 The website is mobile friendly
Contains jQuery errors?	🟢 The website does not contain jQuery errors
Is SSL secured?	🟢 The website is SSL secured
Contains spelling errors?	🟢 The website does not contain spelling errors

Certificate of Proof

🟡 Not KYC verified by Coinsult

ViFoxCoin

Audited by Coinsult.net

Date: 24 November 2025

✔ Advanced Manual Smart Contract Audit

Disclaimer

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Coinsult is not responsible if a project turns out to be a scam, rug-pull or honeypot. We only provide a detailed analysis for your own research.

Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

The information provided in this audit is for informational purposes only and should not be considered investment advice. Coinsult does not endorse, recommend, support or suggest to invest in any project.

Coinsult can not be held responsible for when a project turns out to be a rug-pull, honeypot or scam.

Coinsult

End of report
Smart Contract Audit

CoinsultAudits

info@coinsult.net

coinsult.net