QuillAudits

AUDIT REPORT

May 2025

For

CIFD

HEXchange it!

Table of Content

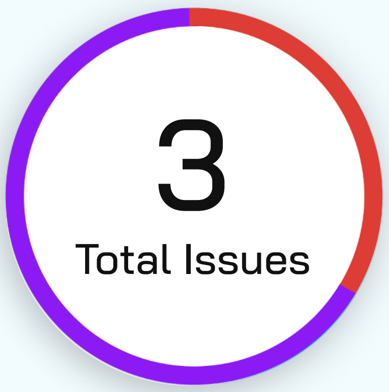
Executive Summary

Project Name	CFID Token
Protocol Type	Token
Project URL	https://www.cifdaq.io/
Overview	An ERC20 token with a 4-year founder vesting schedule and role-based minting. Owner can pause/ unpause, has MINTER_ROLE. Only mints initial tokens if deployed on correct chain
Audit Scope	The scope of this Audit was to analyze the CFID Token Smart Contracts for quality, security, and correctness.
Source Code Link	Zip File Provided by CIFD Team
Contracts in Scope	Cifdsharesv5.sol
Language	Solidity
Blockchain	EVM
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	May 18 2025 - May 26 2025
Updated Code Received	30th May 2025
Review 2	30th May 2025 - 4th May 2025
Fixed In	https://drive.google.com/file/d/14SritnC9IDiKxD_6TejpeXmuMxUxjCKt/view?usp=drive_link

Verify the Authenticity of Report on QuillAudits Leaderboard:

https://www.quillaudits.com/leaderboard

Number of Issues per Severity



🟥	Critical	0 (0%)
🟥	High	1 (33%)
🟧	Medium	0 (0%)
🟨	Low	0 (0%)
🟪	Informational	2 (66%)

Severity

Issues

	🟥 Critical	🟥 High	🟧 Medium	🟨 Low	🟪 Informational
Open	0	0	0	0	0
Acknowledged	0	0	0	0	0
Partially Resolved	0	0	0	0	0
Resolved	0	**1**	0	0	**2**

Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Vesting Schedule can be exploited by unlocking multiple times	High	Resolved
2	Inconsistent Token Name in Permit	Informational	Resolved
3	Remove unused code	Informational	Resolved

Checked Vulnerabilities

- ✓ Access Management
- ✓ Arbitrary write to storage
- ✓ Centralization of control
- ✓ Ether theft
- ✓ Improper or missing events
- ✓ Logical issues and flaws
- ✓ Arithmetic Computations Correctness
- ✓ Race conditions/front running
- ✓ SWC Registry
- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Malicious libraries
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ ERC's conformance
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls

- ✓ **Missing Zero Address Validation**
- ✓ **Private modifier**
- ✓ **Revert/require functions**
- ✓ **Multiple Sends**
- ✓ **Using suicide**
- ✓ **Using delegatecall**
- ✓ **Upgradeable safety**
- ✓ **Using throw**
- ✓ **Using inline assembly**
- ✓ **Style guide violation**
- ✓ **Unsafe type inference**
- ✓ **Implicit visibility level**

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code
- Use of best practices
- Code documentation and comments, match logic and expected behavior
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper
- Implementation of ERC standards
- Efficient use of gas
- Code is safe from re-entrancy and other vulnerabilities

The following techniques, methods, and tools were used to review all the smart contracts:

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms Used for Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity Statistic Analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are five levels of severity, and each of them has been explained below.

🟥 Critical: Immediate and Catastrophic Impact

Critical issues are the ones that an attacker could exploit with relative ease, potentially leading to an immediate and complete loss of user funds, a total takeover of the protocol's functionality, or other catastrophic failures. Critical vulnerabilities are non-negotiable; they absolutely must be fixed.

🟥 High (H): Significant Risk of Major Loss or Compromise

High-severity issues represent serious weaknesses that could result in significant financial losses for users, major malfunctions within the protocol, or substantial compromise of its intended operations. While exploiting these vulnerabilities might require specific conditions to be met or a moderate level of technical skill, the potential damage is considerable. These findings are critical and should be addressed and resolved thoroughly before the contract is put into the Mainnet.

🟧 Medium (M): Potential for Moderate Harm Under Specific Circumstances

Medium-severity bugs are loopholes in the protocol that could lead to moderate financial losses or partial disruptions of the protocol's intended behavior. However, exploiting these vulnerabilities typically requires more specific and less common conditions to occur, and the overall impact is generally lower compared to high or critical issues. While not as immediately threatening, it's still highly recommended to address these findings to enhance the contract's robustness and prevent potential problems down the line.

🟧 Low (L): Minor Imperfections with Limited Repercussions

Low-severity issues are essentially minor imperfections in the smart contract that have a limited impact on user funds or the core functionality of the protocol. Exploiting these would usually require very specific and unlikely scenarios and would yield minimal gain for an attacker. While these findings don't pose an immediate threat, addressing them when feasible can contribute to a more polished and well-maintained codebase.

🟪 Informational (I): Opportunities for Improvement, Not Immediate Risks

Informational findings aren't security vulnerabilities in the traditional sense. Instead, they highlight areas related to the clarity and efficiency of the code, gas optimization, the quality of documentation, or adherence to best development practices. These findings don't represent any immediate risk to the security or functionality of the contract but offer valuable insights for improving its overall quality and maintainability. Addressing these is optional but often beneficial for long-term health and clarity.

Severity Matrix

Impact

Likelihood	🟥 High	🟧 Medium	🟨 Low
🟥 High	Critical	High	Medium
🟧 Medium	High	Medium	Low
🟨 Low	Medium	Low	Low

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

Types of Issues

Open Security vulnerabilities identified that must be resolved and are currently unresolved.	**Resolved** These are the issues identified in the initial audit and have been successfully fixed.
Acknowledged Vulnerabilities which have been acknowledged but are yet to be resolved.	**Partially Resolved** Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

High Severity Issues

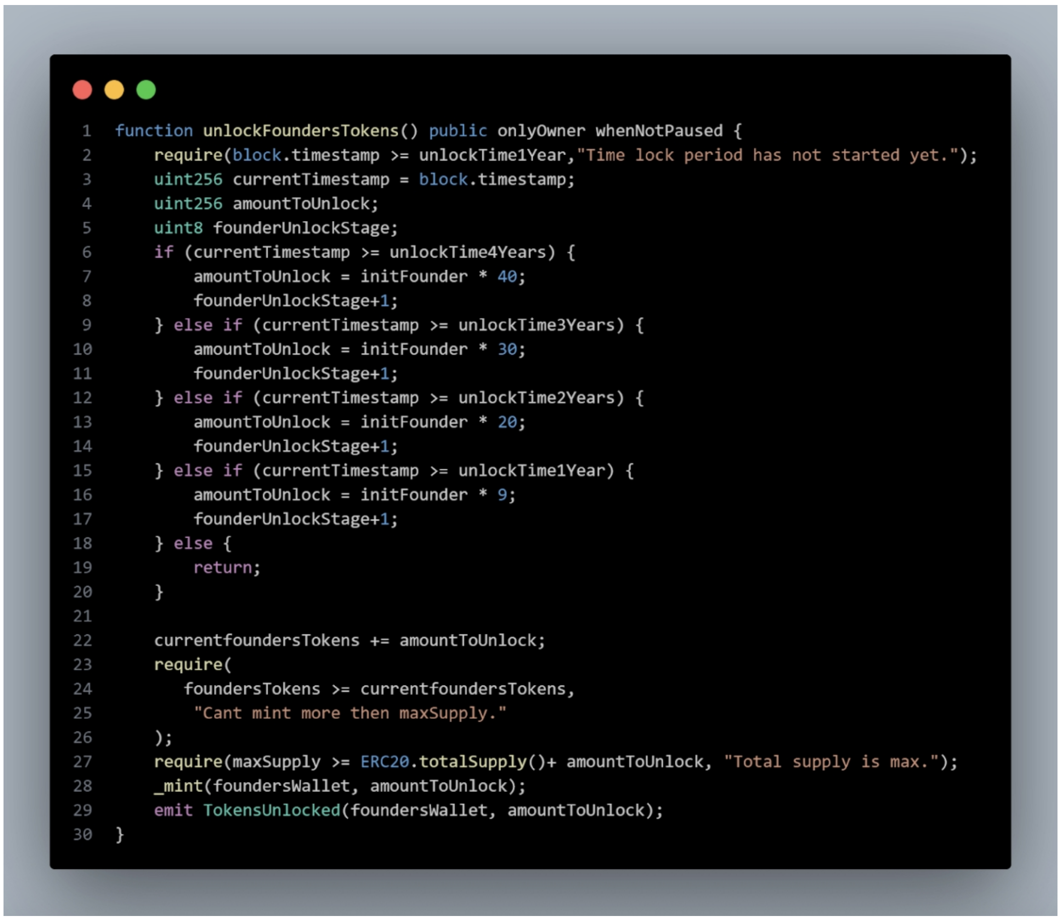
1. Vesting Schedule can be exploited by unlocking multiple times

Resolved

Path
Cifdsharesv5.sol

Function Name
unlockFoundersTokens()

Description
The unlockFoundersTokens() function lacks proper tracking of vesting stages, allowing the contract owner to repeatedly call the function and mint the same vesting amount multiple times at each unlock period. This critical vulnerability stems from the absence of any mechanism to record which vesting stages have already been claimed.



```
function unlockFoundersTokens() public onlyOwner whenNotPaused {
    require(block.timestamp >= unlockTime1Year,"Time lock period has not started yet.");
    uint256 currentTimestamp = block.timestamp;
    uint256 amountToUnlock;
    uint8 founderUnlockStage;
    if (currentTimestamp >= unlockTime4Years) {
        amountToUnlock = initFounder * 40;
        founderUnlockStage+1;
    } else if (currentTimestamp >= unlockTime3Years) {
        amountToUnlock = initFounder * 30;
        founderUnlockStage+1;
    } else if (currentTimestamp >= unlockTime2Years) {
        amountToUnlock = initFounder * 20;
        founderUnlockStage+1;
    } else if (currentTimestamp >= unlockTime1Year) {
        amountToUnlock = initFounder * 9;
        founderUnlockStage+1;
    } else {
        return;
    }

    currentfoundersTokens += amountToUnlock;
    require(
       foundersTokens >= currentfoundersTokens,
        "Cant mint more then maxSupply."
    );
    require(maxSupply >= ERC20.totalSupply()+ amountToUnlock, "Total supply is max.");
    _mint(foundersWallet, amountToUnlock);
    emit TokensUnlocked(foundersWallet, amountToUnlock);
}
```

Impact

This vulnerability allows the contract owner to mint significantly more tokens than intended by the vesting schedule, potentially minting the entire 100M founder allocation immediately after year 1, completely bypassing the 4-year vesting period. Let's say it's been 2 years since deployment.

First Call:

- currentTimestamp >= unlockTime2Years
- amountToUnlock = 20M
- currentfoundersTokens = 0 + 20M = 20M
- Check: 100M >= 20M?
- Mints 20M tokens

Second Call (immediately after):

- currentTimestamp >= unlockTime2Years? (time hasn't changed)
- amountToUnlock = 20M (same calculation)
- currentfoundersTokens = 20M + 20M = 40M
- Check: 100M >= 40M?
- Mints another 20M tokens

Third Call:

- currentTimestamp >= unlockTime2Years?
- Process repeats...

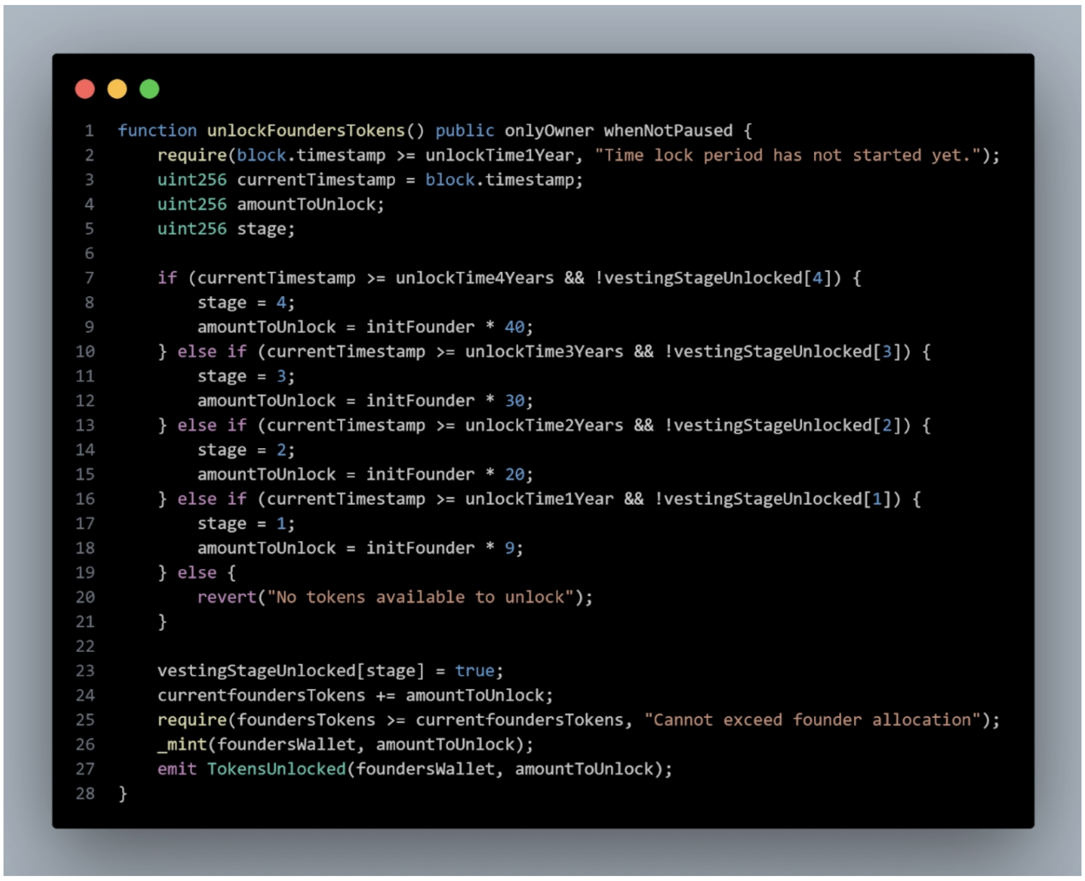
Nothing records that year 2 was already processed. The only thing stopping infinite minting is when currentfoundersTokens exceeds foundersTokens

Likelihood

High - The exploit is trivial to execute, requiring only repeated function calls by the owner. No special conditions or external factors are needed.

Recommendation

Implement a proper vesting stage tracking mechanism:



```
function unlockFoundersTokens() public onlyOwner whenNotPaused {
    require(block.timestamp >= unlockTime1Year, "Time lock period has not started yet.");
    uint256 currentTimestamp = block.timestamp;
    uint256 amountToUnlock;
    uint256 stage;

    if (currentTimestamp >= unlockTime4Years && !vestingStageUnlocked[4]) {
        stage = 4;
        amountToUnlock = initFounder * 40;
    } else if (currentTimestamp >= unlockTime3Years && !vestingStageUnlocked[3]) {
        stage = 3;
        amountToUnlock = initFounder * 30;
    } else if (currentTimestamp >= unlockTime2Years && !vestingStageUnlocked[2]) {
        stage = 2;
        amountToUnlock = initFounder * 20;
    } else if (currentTimestamp >= unlockTime1Year && !vestingStageUnlocked[1]) {
        stage = 1;
        amountToUnlock = initFounder * 9;
    } else {
        revert("No tokens available to unlock");
    }

    vestingStageUnlocked[stage] = true;
    currentfoundersTokens += amountToUnlock;
    require(foundersTokens >= currentfoundersTokens, "Cannot exceed founder allocation");
    _mint(foundersWallet, amountToUnlock);
    emit TokensUnlocked(foundersWallet, amountToUnlock);
}
```

Informational Issues

2. Inconsistent Token Name in Permit

Resolved

Path
Cifdsharesv5.sol

Function Name
constructor()

Description
The ERC20Permit constructor uses “cifdsharesv2” while the token name is “cifdsharesv5”, creating an inconsistency that could cause confusion.

Recommendation
Sync both the versions

3. Remove unused code

Resolved

Path
Cifdsharesv5.sol

Description
The function declares a local variable founderUnlockStage and but the variable is never used.

Recommendation
Remove the unused variable and statements, or properly implement stage tracking.

Functional Tests

Some of the tests performed are mentioned below:

- ✓ Should revert if unlocking is done prior
- ✓ Should revert when paused
- ✓ Should revert with zero address inputs
- ✓ Max Supply should never be exceeded

Automated Tests

No major issues were found. Some false-positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Threat Model

Contract	Function	Threats
Cifdsharesv5.sol	unlockFoundersTokens()	**Inputs** • No external inputs - function operates based on block timestamp and contract state. **Internal State Dependencies** • block.timestamp • **Control:** None (controlled by blockchain). • **Constraints:** Must be >= unlockTime1Year to execute. • **Impact:** Determines which vesting tier is eligible. • currentfoundersTokens • **Control:** Modified by this function only. • **Constraints:** Must not exceed foundersTokens (100M). • **Impact:** Tracks cumulative founder tokens minted. **Branches and Code Coverage** • Should unlock 9% of founder allocation after 1 year. • Should unlock 20% of founder allocation after 2 years. • Should unlock 30% of founder allocation after 3 years.

Contract	Function	Threats
		<ul><li>Should unlock 40% of founder allocation after 4 years.</li><li>Should mint tokens to founders wallet. Should emit</li><li>TokensUnlocked event. Should update currentfoundersTokens tracking.</li></ul><ul><li>currentfoundersTokens<ul><li>Should not allow unlocking before 1 year.</li><li>Should not allow unlocking more than foundersTokens allocation.</li><li>Should not work when paused.</li><li>Currently allows multiple unlocks at same tier (critical bug).</li><li>Currently only unlocks single tier instead of cumulative (logic bug).</li><li>Dead code with unused founderUnlockStage variable.</li><li>Only callable by owner (centralization risk).</li></ul></li></ul>

Closing Summary

In this report, we have considered the security of CFID Token. We performed our audit according to the procedure described above.

Issues of High and Informational severity were found , CFID team resolved them all

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

7+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

May 2025

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com