QuillAudits

AUDIT REPORT

December, 2024

For

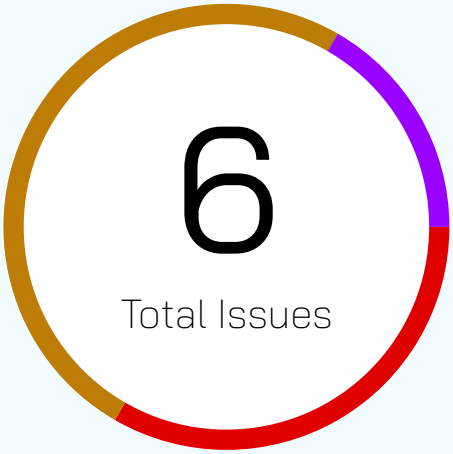
Table of Content

Executive Summary

Project name	CIFD
Overview	CIFD contract defines a custom token called CIFDToken, which inherits from OpenZeppelin's ERC20 and Ownable contracts. The main purpose appears to be a token distribution with time-locked releases to different wallets. Vault contract implements a simple time-lock mechanism for deposits and withdrawals. It has two main functions: deposit and withdraw. unlockTime: The timestamp when funds can be withdrawn. owner: The address of the owner who can initiate withdrawals. Two events: Deposit and Withdrawal, emitted when funds are added or removed. Functions deposit(): Allows anyone to send ETH to the contract, emitting a Deposit event. withdraw(): Can only be called by the owner after the unlockTime has passed. It transfers all funds to the owner and emits a Withdrawal event.
Project URL	www.cifdaq.io
Audit Scope	The scope of this Audit was to analyze the CFID Token Smart Contracts for quality, security, and correctness.
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Contracts in Scope	https://github.com/Theo1016/CIFDToken/tree/main/contracts Branch: Main Contracts:- contracts/CIFDToken.sol contracts/Vault.sol contracts/DeterministicDeployFactory.sol

Commit Hash	0226ea92bb7a654a37ca06a70132b36babd11a3c
Language	Solidity
Blockchain	EVM
Review 1	2nd December 2024 - 4th December 2024
Updated Code Received	10th December 2024
Review 2	10th December 2024
Fixed In	7f9ad6fb9b7149714b76795f1aadd04302182d92

Number of Issues per Severity



High	2 (33.33%)
Medium	3 (50.00%)
Low	0 (0.00%)
Informational	1 (16.67%)

Severity

Issues	High	Medium	Low	Informational
Open	0	0	0	0
Resolved	**2**	0	0	**1**
Acknowledged	0	**3**	0	0
Partially Resolved	0	0	0	0

Checked Vulnerabilities

- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ DoS with Block Gas Limit
- ✓ Transaction-Ordering Dependence
- ✓ Use of tx.origin
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Balance Equality
- ✓ Byte Array
- ✓ Transfer Forwards All Gas
- ✓ ERC20 API Violation
- ✓ Compiler Version Not Fixed
- ✓ Redundant Fallback Function
- ✓ Send Instead of Transfer
- ✓ Style Guide Violation
- ✓ Unchecked External Call
- ✓ Unchecked Math
- ✓ Unsafe Type Inference
- ✓ Implicit Visibility Level
- ✓ Access Management
- ✓ Arbitrary Write to Storage
- ✓ Centralization of Control
- ✓ Ether Theft
- ✓ Improper or Missing Events
- ✓ Arithmetic Computations Correctness
- ✓ Logical Issues and Flaws
- ✓ Race Conditions/Front Running
- ✓ SWC Registry
- ✓ Malicious Libraries
- ✓ Address Hardcoded
- ✓ Divide Before Multiply

- ☑ Integer Overflow/Underflow
- ☑ Dangerous Strict Equalities
- ☑ Tautology or Contradiction
- ☑ Return Values of Low-Level Calls
- ☑ Missing Zero Address Validation
- ☑ ERC's Conformance
- ☑ Private Modifier
- ☑ Revert/Require Functions
- ☑ Multiple Sends
- ☑ Using Suicide
- ☑ Using Delegatecall
- ☑ Upgradeable Safety
- ☑ Using Throw
- ☑ Using Inline Assembly

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 statistical analysis.

Types of Severity

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

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational Issues

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

Security vulnerabilities identified that must be resolved and are currently unresolved.

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

Misuse of Deposit

Resolved

Path

Vault.sol

Function

deposit()

Description

Anyone can call the deposit() function and emit the Deposit event, even without sending Ether. This could create misleading events in the logs.

Recommendation

Ensure msg.value > 0 before emitting the event

```
function deposit() public payable {
require(msg.value > 0, "Must send some Ether");
emit Deposit(msg.value, block.timestamp);
}
```

Incorrect Calculation of Unlock Amount

Resolved

Path

CFIDToken.sol

Function

unlockFoundersTokens()

Description

In unlockFoundersTokens, the calculation for amountToUnlock is based on a fixed multiplier of init-Founder. However, it does not track already unlocked tokens, leading to potential double minting or excess supply beyond foundersTokens

Recommendation

Introduce a variable to track the already unlocked tokens and ensure no over-minting occurs: Replace the use of initFounder with foundersTokens in the unlockFoundersTokens() function to ensure correct token distribution as per the vesting schedule.

```
uint256 public unlockedTokens;

function unlockFoundersTokens() public onlyOwner {
require(block.timestamp >= unlockTime1Year, "Time lock period has not started yet.");
require(maxSupply > totalSupply(), "Max supply reached.");

uint256 currentTimestamp = block.timestamp;
uint256 totalUnlockable;

// if logic here ...

uint256 amountToUnlock = totalUnlockable - unlockedTokens;
require(amountToUnlock > 0, "No tokens available to unlock.");

unlockedTokens += amountToUnlock;
 _mint(foundersWallet, amountToUnlock);
emit TokensUnlocked(foundersWallet, amountToUnlock);
}
```

Medium Severity Issues

Unlock Time Misconfiguration

Acknowledged

Path

Vault.sol

Description

unlockTime is set during deployment but cannot be updated. This makes the contract inflexible, as the owner might want to extend the unlock time.

Recommendation

Allow the owner to update the unlock time if necessary

```
function updateUnlockTime(uint _newUnlockTime) public {
require(msg.sender == owner, "Only the owner can update the unlock time");
require(_newUnlockTime > block.timestamp, "Unlock time must be in the future");
unlockTime = _newUnlockTime;
}
```

Lack of Ownership Transfer Function

Acknowledged

Description

The owner cannot transfer ownership to another address, which limits the contract's usability if ownership needs to change.

Recommendation

Add a function to transfer ownership

```
function transferOwnership(address payable newOwner) public {
require(msg.sender == owner, "Only the owner can transfer ownership");
require(newOwner != address(0), "New owner cannot be the zero address");
owner = newOwner;}
}
```

Hardcoded Unlock Durations

Acknowledged

Path

CFIDToken.sol

Description

The unlockTime1Year, unlockTime2Years, etc., are hardcoded relative to the contract deployment time. This reduces flexibility if timeframes need adjustment.

Recommendation

Allow the unlock times to be configurable via the constructor.

Informational Severity Issues

Improved Events

Resolved

Path

CFIDToken.sol

Description

The TokensUnlocked event does not specify which unlock stage was triggered.

Recommendation

Include a parameter to indicate the unlocking stage

event TokensUnlocked(address beneficiary, uint256 amount, uint256 stage);

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.

Closing Summary

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

Few issues of High,Medium and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in CIFD smart contract. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of CIFD smart contract. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the CIFD to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

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.

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

Follow Our Journey

AUDIT REPORT

December, 2024

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com