SMART CONTRACTS REVIEW

zokyo

July 25th 2024 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passed a security audit.

ZOKYO AUDIT SCORING CYBER FINANCE

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 1 High issue: 1 resolved = 0 points deducted
- 1 Medium issue: 1 acknowledged = - 3 points deducted
- 3 Low issues: 1 resolved and 2 acknowledged = - 2 points deducted
- 3 Informational issues: 1 acknowledged and 2 resolved = 0 points deducted

Thus, 100 - 3 - 2 = 95

TECHNICAL SUMMARY

This document outlines the overall security of the Cyber Finance smart contract/s evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Cyber Finance smart contract/s codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the review. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract/s but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Cyber Finance team put in place a bug bounty program to encourage further active analysis of the smart contract/s.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Cyber Finance repository:
Repo: https://github.com/cyberfin-xyz/contract/commit/86542aedb912e7f56640f746c20c5c7274119f9c

Last commit -5d9cdeb9c292ad93b040f47693511d27e1e955d6

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- CyberFinance.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Cyber Finance smart contract/s. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01 Due diligence in assessing the overall code quality of the codebase.

02 Cross-comparison with other, similar smart contract/s by industry leaders.

03 Thorough manual review of the codebase line by line.

Executive Summary

The Zokyo team has performed a security audit of the provided codebase. Detailed findings from the audit process are outlined in the "Complete Analysis" section.

The Cyber Finance contract is designed to manage and distribute contest rewards in the form of ERC20 tokens. It allows the contract owner to allocate rewards to participants, and participants can claim their rewards when ready.

Cyber Finance includes a mapping to track claimable balances for each user and token pair, allowing the owner to adjust these balances through functions to increase or decrease allocations. The contract supports administrative operations like pausing and unpausing the contract, ensuring control over operational states.

The owner has exclusive rights to withdraw tokens from the contract, either partially or fully. Users can claim their tokens via the claim function, which checks and transfers the claimable balance. The contract also provides structured event logging for actions like balance adjustments and withdrawals, facilitating transparency and operations monitoring.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Cyber Finance team and the Cyber Finance team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Non-Standard ERC20 tokens could be locked in the contract	High	Resolved
2	Centralization Risk due to overpowered owner	Medium	Acknowledged
3	PUSH0 Opcode is Incompatible with some Chains	Low	Acknowledged
4	Owner Can Renounce Ownership While System is Paused	Low	Acknowledged
5	Contract Balance Insufficiency Prevents User Withdrawals	Low	Resolved
6	Front Running Vulnerability in decreaseClaimable Function	Informational	Acknowledged
7	Redundant Modifier Invocation in Withdraw Functions	Informational	Resolved
8	Public Functions Could Be Declared External	Informational	Resolved

HIGH-1 | RESOLVED

Non-Standard ERC20 tokens could be locked in the contract

The ERC20 `transfer` function is used to transfer `rewardToken` tokens. However, some ERC20 tokens, such as USDT, BNB, and OMG, do not return a boolean. This can cause the `require(success, "Token transfer failed")` check to always fail, potentially resulting in tokens being locked in the contract.

Recommendation:

To ensure compatibility with all ERC20 tokens and to handle transfer failures properly, it is recommended to use the SafeERC20 library from OpenZeppelin. This library provides a safe transfer function that properly handles tokens that do not return a value.

Replace:

```
bool success = IERC20(rewardToken).transfer(owner(), amount);
```

With:

```
SafeERC20.safeTransfer(IERC20(rewardToken), owner(), amount);
```

MEDIUM-1 | ACKNOWLEDGED

Centralization Risk due to overpowered owner

The `CyberFinance` contract introduces a significant centralization risk due to its reliance on the owner for critical functions, including pausing/unpausing the contract, increasing/decreasing claimable balances, and withdrawing tokens. This centralized control can lead to potential abuse or single points of failure, particularly if the owner account is compromised or if the owner acts maliciously.

Recommendation:

To mitigate centralization risks, consider implementing multisig wallets or time-locked functions for critical operations.
Comment: The client stated that the project is planned to set up a multisig for the owner wallet to mitigate the centralization.

LOW-1 | ACKNOWLEDGED

`PUSH0` Opcode is Incompatible with some Chains

The `CyberFinance` contract is written using Solidity version `0.8.21`, which introduces the `push0` opcode. This opcode is not supported by all chains, especially those not compatible with the Shanghai hardfork. Deploying this contract on such incompatible chains could lead to deployment failures.

This means that the produced bytecode won't be compatible with the chains that don't yet support the Shanghai hard fork. This could also become a problem if different versions of Solidity are used to compile contracts for different chains. The differences in bytecode between versions can impact the deterministic nature of contract addresses.

Recommendation:
To ensure broader compatibility and prevent deployment issues, it is recommended to either roll back the Solidity version or hardcode the EVM version to "paris" in the Foundry configuration file (`foundry.toml`).

LOW-2 | ACKNOWLEDGED

Owner Can Renounce Ownership While System is Paused

The `CyberFinance` contract inherits from `Ownable2Step` and includes `Pausable` functionality. However, the current implementation allows the owner to renounce ownership even while the system is paused. This can lead to a scenario where the contract is left in an unusable state and the funds could be locked there forever, as no one would have the authority to unpause the contract and resume normal operations.

Recommendation:
Implement a check to prevent the owner from renouncing ownership while the contract is paused. This ensures that the system remains manageable and prevents it from being locked in a paused state indefinitely.

LOW-3 | RESOLVED

Contract Balance Insufficiency Prevents User Withdrawals

The `claim` function currently restricts withdrawals if the contract's balance of the reward token is less than the user's claimable balance. This may lead to user frustration, as they are unable to withdraw any portion of their claimable assets under these conditions. A scenario where users are unable to claim their rewards due to contract balance constraints undermines user trust and satisfaction.

Recommendation:
To address this issue, modify the `claim` function to allow users to withdraw whatever portion of their claimable balance is available in the contract. If the contract's balance is less than the user's claimable balance, the function should allow the user to withdraw the available balance and subsequently deduct the withdrawn amount from the user's claimable balance. This ensures that users can at least partially receive their assets and improves user experience by mitigating frustrations related to withdrawal failures.

INFORMATIONAL-1 | ACKNOWLEDGED

Front Running Vulnerability in decreaseClaimable Function

The `decreaseClaimable` function, which can only be called by an admin, is vulnerable to front running by users who call the `claim` function. This allows the users to drain the claimable balance before it is decreased by the `decreaseClaimable` function. This behavior can potentially lead to unexpected fund depletion or malicious draining of resources.

Recommendation:
To mitigate this vulnerability, consider implementing a two-step claim process. Introduce a `commitClaim` function where the user commits to a planned claim as the first step. The user would then finalize the claim after a certain number of blocks have passed and before a specified deadline block. This mechanism would significantly hinder opportunistic claims based on admin actions, thereby reducing the risk of front-running exploits.

INFORMATIONAL-2 | RESOLVED

Redundant Modifier Invocation in Withdraw Functions

The `onlyOwner` modifier is invoked twice in `withdrawBalance` and `withdraw` functions, leading to inefficiency in the smart contract. Both functions perform a similar check that could be consolidated.

Recommendation:
To improve the efficiency, refactor the withdraw logic into an internal function. Both `withdraw` and `withdrawBalance` can then call this internal function, ensuring that the modifiers are applied only once. This reduces redundancy and improves code readability and efficiency.

INFORMATIONAL-3 | RESOLVED

Public Functions Could Be Declared External

In Solidity, functions can be declared as `public` or `external`. While both visibility specifiers allow functions to be called from outside the contract, `external` functions are generally more gas-efficient when called externally. This is because external functions use a lower amount of gas due to optimized calldata handling.

In the `CyberFinance` contract, several functions that are intended to be called only from outside the contract are declared as `public`. These functions could be declared as `external` to optimize gas usage.

Recommendation:
Review the contract to identify functions that are intended to be called only externally and declare them as `external` instead of `public`.

	CyberFinance.sol
Reentrance	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

We are grateful for the opportunity to work with the Cyber Finance team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Cyber Finance team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo