CYBRO × PESSIMISTIC

SECURITY ANALYSIS

by Pessimistic

This report is public

October 28, 2024

ABSTRACT

In this report, we consider the security of smart contracts of CYBRO project. Our task is to find and describe security issues in the smart contracts of the platform.

DISCLAIMER

The audit does not give any warranties on the security of the code. A single audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, a security audit is not investment advice.

SUMMARY

In this report, we considered the security of CYBRO smart contracts. We described the audit process in the section below.

The audit showed the Inflation attack critical issue. The audit also revealed several issues of medium severity: Blocked interactions by one of the external troubled pools, Discrepancy with documentation, Inconsistency between withdrawal fee calculations, Inconsistency between perfomance fee calculations, Project roles and Separate logic for removing pools and redeeming assets from them. Moreover, several low-severity issues were found.

The overall code quality is mediocre.

After the initial audit, the codebase was updated. The developers commented the Inflation attack issue of critical severity and fixed Inconsistency between perfomance fee calculations issue. Also the developers left a comment about other mentioned issues:

> "Only C01 and M04 have been fixed, the rest are acknowledged. We ourselves plan to make a non-removable deposit when the contract is deployed. This way we cover the vulnerability C01."

Inflation attack issue has not been fixed at this codebase update.

The number of tests increased. All tests successfully passed.

GENERAL RECOMMENDATIONS

We recommend fixing the remaining issues and improving code readability.

PROJECT OVERVIEW

Project description

For the audit, we were provided with CYBRO project on a public GitHub repository, commit d4ff8b4c32d1cce6ca45b2f0b6a54a9743b99a74.

The scope of the audit included:

- **OneClickLending.sol**;
- **FeeProvider.sol**;
- **ILendingPool.sol**;
- **IFeeProvider**.

The documentation for the project included a private Notion document.

All 19 tests pass successfully. The code coverage is 81,07%.

The total LOC of audited sources is 352.

Codebase update #1

After the initial audit, the developers provided an updated version of the CYBRO project code: commit f3324ef720eaba183ddddde07ad618a32197d03. In this update, Inconsistency between perfomance fee calculations issue was fixed. No new issues were found.

All 26 tests passed. The code coverage is 85,03%.

AUDIT PROCESS

We started the audit on October 17, 2024 and finished on October 21, 2024.

We inspected the materials provided for the audit. We performed preliminary research and specified those parts of the code and logic that require additional attention during the audit:

- Whether there is no unexpected possibility to withdraw assets from the vault;
- The integrations with external pools;
- Whether the shares and their price calculations are correct;
- Whether the roles' influence does not affect the contract much;
- Standard Solidity checks.

During the work, we stayed in touch with the developers and discussed confusing or suspicious parts of the code.

We manually analyzed all the contracts within the scope of the audit and checked their logic.

We scanned the project with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules;
- Semgrep rules for smart contracts. We also sent the results to the developers in the text file.

We ran tests and calculated the code coverage.

We combined in a private report all the verified issues we found during the manual audit or discovered by automated tools.

After the initial audit, on October 25, 2024 the developers provided us with an updated version of the code. In this update, they added tests, commented and fixed several issues from our report.

We reviewed the updated codebase and scanned it with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules.

We did not find any new issues and all tests pass in this version of the code.

Finally, we updated the report.

MANUAL ANALYSIS

The contracts were completely manually analyzed, and their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

C01. Inflation attack (commented)

The inflation attack can be applied to line 160 of the `OneClickLending.deposit` function.

The example:

- The first malicious user calls the `deposit` function and sends 1 underlying token. As a result, `totalSupply = 1`, `totalAssetsBefore = 1`;
- Then, the second user wants to deposit 3 underlying tokens;
- Malicious user can front-run the transaction of the second user and transfer 6 `poolAddress` tokens to contract;
- The amount of shares for the second user is equal to 0, as `totalSupply = 1`, `totalAssetsBefore = 1 + 6`, `increase = 3`, and `totalSupply() * increase / totalAssetsBefore < 1`;
- It can be repeated multiple times;
- Malicious user calls the `redeem` function, burns his 1 share, and withdraw all underlying tokens from the vault.

> `Comment from the developers:`
> **We ourselves plan to make a non-removable deposit when the contract is deployed. This way we cover the vulnerability C01.**

Medium severity issues

Medium severity issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. Blocked interactions by one of the external troubled pools

The **OneClickLending** contract allows interaction with external pools only when all of the pools works properly. In case one of the external pools returns an error, it will be impossible to deposit or redeem assets.

M02. Discrepancy with documentation

There are several places in the **OneClickLending** contract that do not align with the description in the documentation:

- The `pause` and `unpause` functions have the wrong access restriction role;
- There is no function to compute a clear equity in the **OneClickLending** contract.

M03. Inconsistency between withdrawal fee calculations

The balance of deposited assets is calculated based on `account`, while the fee is calculated based on `msg.sender` at lines 390 and 393 in the `OneClickLending.quoteWithdrawalFee` function. If `msg.sender` and `account` are different addresses, the value may be calculated incorrectly.

M04. Inconsistency between perfomance fee calculations (fixed)

The perfomance fee is calculated based on `msg.sender` which is the `MANAGER` at line 302 in the `collectPerformanceFee` function. But at line 532 in the `_applyPerformanceFee` function the perfomance fee is calculated based on `msg.sender` which is the user-depositor. This may result in the depositor being charged a different amount of fees if user and `MANAGER` have different performance fee rates.

> The issue has been fixed and is not present in the latest version of the code.

M05. Project roles

The **OneClickLending** contract includes several roles with varying degrees of influence on its behavior. Consider designing contract in a trustless manner or implementing proper key management, e.g., setting up a multisig to avoid scenarios that can lead to undesirable consequences for the project and its users, e.g., if admin's private keys become compromised.

- `DEFAULT_ADMIN`:
  - Can pause/unpause `deposit` function;
  - Can withdraw assets from the vault;
  - After the contract is deployed, the `DEFAULT_ADMIN` is not renounced;
  - **(fixed)** After the contract is deployed, initial `DEFAULT_ADMIN` has all the roles.
- `STRATEGIST` can:
  - Add lending pools;
  - Remove lending pools.
- `MANAGER` can:
  - Set lending shares;
  - Rebalance assets using `rebalance` or `rebalanceAuto` functions;
  - Collect performance fee.
- `MANAGER` and `STRATEGIST` together can:
  - Add custom pool to steal assets by rebalancing.
- `DEFAULT_ADMIN` and `STRATEGIST` together can:
  - Remove pool from `lendingPoolAddresses` and withdraw shares of the removed pool using `withdrawFunds` function.
- Also external **FeeProvider** contract can:
  - Exact users' whole `deposit`/`redeem` assets by 100% fee;
  - Block `deposit`/`redeem` by setting fee above 100%.

M06. Separate logic for removing pools and redeeming assets from them

The `removeLendingPools` function removes pool from the `lendingPoolAddresses` list, resetting its `lendingShares` and reduces `totalLendingShares`. However, the assets are not withdrawn from the removed pool at the same time. As a result, most important functions might be executed incorrectly, if the assets were not fully withdrawn before the pool was removed. This concerns `totalAssets` and `sharePrice` functions and all other functions that use it for calculations.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Uneven distribution of small deposited assets

In case the user deposits an `assets` amount less than `totalLendingShares` / `lendingShares[poolAddress]` at line 471 in `_deposit` function of the **OneClickLending** contract, `amountToDeposit` will be equal to 0 until the last iteration of the `for` loop. This will result in all the assets being deposited into one pool, regardless of its `lendingShares[poolAddress]` weight.

L02. Uneven distribution of deposited assets through unspecified lendingShares

In case `lendingShares` are not specified and `totalLendingShares` also equals 0, all the assets are being deposited into one `poolAddress` pool at lines 468-476 in the `_deposit` function. Even though the pool has 0 `lendingShares[poolAddress]` weight.

L03. Unchangeable gas mode

Contracts deployed on Blast network can collect spent gas fees setting their gasMode to claimable.

L04. Minimal assets amount check absent

Consider adding `minAssets` parameter during withdrawal, to protect users from losing assets due to unexpected behaviour of external pools.

L05. Duplicate code

We recommend using `_computeDeviantion` function at line 264 in the `rebalanceAuto` to increase code readability.

L06. Rebalance improvement check absent

We recommend checking that new `deviation` of the `to` pool is better than previous one in the `rebalance` function to exclude meaningless or malicious rebalancing.

L07. Safe type converting absent

We recommend using **SafeCast** library at lines 264, 275, 276 and 447 in `_computeDeviation` and `rebalanceAuto` functions according to Solidity best practices.

Notes

N01. Integrations

The project has integrations with CYBRO base vaults that integrated to Aave, Compound and Juice Finance protocols. Any restrictions that may be placed on these pools, such as pause, capped deposits, deprecated pools, or any other, can influence **OneClickLending** vault functionality.

N02. Code readability

We recommend removing following meaningless patterns to increase code readability:

- Converting `address` to `address` type all over the code;
- **(fixed)** Using not overridden `_msgSender()` instead of `msg.sender` at lines 151, 165 and 180 in the `deposit` and `redeem` functions;
- Using `payable address` type for `call` function at line 314 in the `withdrawFunds` function.

N03. Gas consumption

We recommend replacing `public` modifier with `external` and `memory` data location with `calldata` where is possible to decrease gas consumption and increase code readability.

This analysis was performed by Pessimistic:

Oleg Bobrov, Security Engineer
Evgeny Bokarev, Junior Security Engineer
Konstantin Zherebtsov, Business Development Lead
Irina Vikhareva, Project Manager
Alexander Seleznev, CEO

October 28, 2024