QuillAudits

AUDIT REPORT

March, 2025

For

CYBRO

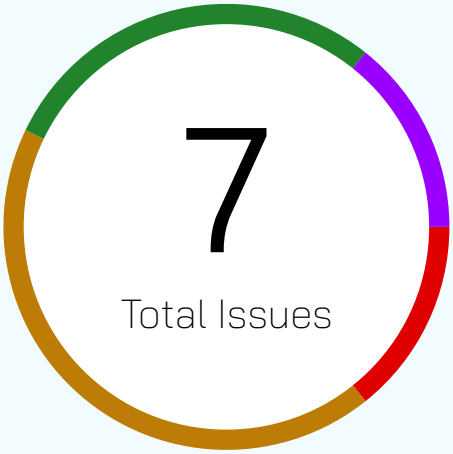
Table of Content

Executive Summary

Project name	Cybro
Overview	CYBRO is a multichain, AI-powered yield aggregator designed to simplify and optimize decentralized finance (DeFi) investments for users
Project URL	https://cybro.io/en
Audit Scope	https://github.com/cybro-io/vault-contracts/pull/12 https://github.com/cybro-io/vault-contracts/pull/13 https://github.com/cybro-io/vault-contracts/pull/14 https://github.com/cybro-io/vault-contracts/pull/15 https://github.com/cybro-io/vault-contracts/pull/16 https://github.com/cybro-io/vault-contracts/pull/17 https://github.com/cybro-io/vault-contracts/pull/18 https://github.com/cybro-io/vault-contracts/pull/19
Contracts in Scop	src/BaseVault.sol src/Exchange.sol src/FeeProvider.sol src/OneClickIndex.sol src/Oracle.sol src/vaults/*.sol src/dex/*.sol src/interfaces/*.sol
Commit Hash	8906044b6942f80ffbd602cc7ff64456f04a7ddb
Language	Solidity
Blockchain	Arbitrum, Base, Blast
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	12th Feb 2025 - 7th Mar 2025

Updated Code Received	12th March 2025
Review 2	13th March 2025 - 17th March 2025
Fixed In	https://github.com/cybro-io/vault-contracts/pull/24

Number of Issues per Severity



High	1 (14.29%)
Medium	3 (42.86%)
Low	2 (28.57%)
Informational	1 (14.29%)

Severity

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

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

- [x] Using inline assembly
- [x] Unsafe type inference
- [x] Style guide violation
- [x] 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 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

RebalanceAuto() does not check for swap on zero amount

Resolved

Path

src/OneClickIndex.sol

Function

rebalanceAuto()

Description

In rebalanceAuto(), It automatically rebalances assets across all lending pools part of what it does is iterate through all lending pool with a for loop, executing redeem on each pool and then swapping it to the pool asset, However it never checks the amount returned on the redeem() to be above 0 before executing the swap as it is a possibility for one of the pool being iterated on to return 0 on redeem and also since the redeem function default is to round down. This unhandled zero amount going into a swap will always revert the entire rebalanceAuto().
Therefore, if one of the pool.redeem() returns zero the entire rebalanceAuto() will fail.

Recommendation

Check that the amount returned on the redeem() is above 0 before executing the swap, else skip the swap for that lending pool

Medium Severity Issues

Small ETH Amounts Are Rounded Down to Zero in _convertETHToUSDB()

Acknowledged

Path

src/Exchange.

Function

_convertETHToUSDB()

Description

The _convertETHToUSDB() function converts ETH to USDB using a price oracle. However, for small input amounts (e.g., 1000 wei), the computation rounds down to zero due to multiple divisions in a fixed-point arithmetic setup.
If volume is too small (e.g., 1000 wei), the result rounds to 0 due to Solidity's integer division truncating decimals.

Recommendation

1. Use mulDiv() from OpenZeppelin's Math library to ensure accurate fixed-point division.
2. Set a Minimum Convertible ETH Amount.

Rewards will forever be unclaimable

Acknowledged

Path

src/vaults/AaveVault.sol

Description

No method for users or protocol to claim the incentives that Aave distributes to suppliers. E.g In Arbitrum, the aave rewards contract, currently this contract is still available for rewards claiming. However, since the contract does not include functionality to claim Aave incentives in its initial design, these rewards will forever be unclaimable. The lack of incentive claiming functionality may discourage users from using the vault contract, as they would miss out on additional earnings.

Recommendation

Add functions that allows users or protocol to claim their incentives

For cases when fee returned is zero _applyPerformanceFee will revert entire parent operations in some tokens.

Resolved

Path

src/BaseVault.sol

Function

_applyPerformanceFee()

Description

Unlike in _applyWithdrawalFee where fee > 0 is first checked before making a transfer, _applyPerformanceFee does not check if fee > 0 before making the performancefee transfer, this will cause operations that makes call to _applyPerformanceFee to revert for some tokens for cases when fee returned is zero.
Transferring performance fee without checking if fee > 0 could cause the entire parent operations to revert for some tokens.

Recommendation

In _applyPerformanceFee() check fee > 0 before transfer, just as in _applyWithdrawalFee()

Low Severity Issues

setFeesForUsers() Called Before getUpdateUser-Fees() Results in Zero Fees for Users

Resolved

Path

src/FeeProvider.sol

Function

setFeesForUsers

Description

The setFeesForUsers() function allows the owner to set custom fees for users, but it does not initialize user fees properly if getUpdateUserFees() has not been called before.
When setFeesForUsers() is executed:
. The function sets the user's fees using Math.min() with the default uninitialized value (0) from _users[user].
. Since userFees.depositFee, userFees.withdrawalFee, and userFees.performanceFee are not initialized, they default to 0.
. As a result, the user's fees are always set to 0, overriding the global fee settings.
. getUpdateUserFees() is expected to initialize fees, but if setFeesForUsers() is called first, the fees will remain 0.

Recommendation

1. Ensure User Fees are Initialized Before Applying Math.min()
a. Before setting fees, check if the user has initialized fees; otherwise, initialize them with the global fee values.
2. Modify setFeesForUsers() to Initialize Fees Properly
3. Also, best to have a check on array lengths, as if arrays are not of the same length. Otherwise, the fee can be set to 0 for those users by default.

Missing lendingPoolAddresses on setLending-Shares() could affect protocol accounting

Resolved

Path

src/OneClickIndex.sol

Function

setLendingShares()

Description

setLendingShares() which is called be manager role does not check if pool is added or has not been removed lendingPoolAddresses by the strategist role before updating the totalLendingShares, if this happens it could affect entire protocol accounting due to additional LendingShares for pool address that are not in the lendingPoolAddresses

Recommendation

check if pool address to include lendingShares to, is added or has not been removed from the lendingPoolAddresses before modifying these values.

Informational Severity Issues

No Restriction on Setting 100% Withdrawal Fee Despite Documentation Limitations

Acknowledged

Path

src/FeeProvider.sol

Function

setFees()

Description

According to the project's documentation, the withdrawal fee cannot be set to 100%, ensuring that users can always withdraw a portion of their funds. However, the setFees() function does not enforce this restriction, allowing the owner to set _withdrawalFee to 100%, effectively preventing users from withdrawing their assets.

Recommendation

Add a maximum limit (e.g., 50%) to _withdrawalFee to align with the documented behavior and prevent abuse.

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 Cybro. We performed our audit according to the procedure described above.

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

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 secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem

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

Follow Our Journey

AUDIT REPORT

March, 2025

For

CYBRO

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com