QuillAudits
Web3 Security

AUDIT REPORT

December, 2024

For

CYBRO

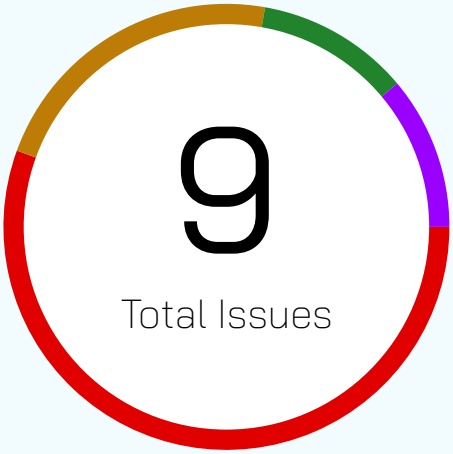
Table of Content

Executive Summary

Project name	Cybro Vaults
Overview	Cybro Vaults create contracts as wrappers around many popular DeFi projects, adding investment fund functionality to them. For example, there's the StarGate protocol. Investors can invest in it independently, then manually claim profits in STG tokens, exchange STG for ETH, and reinvest the ETH back into StarGate. This process requires multiple steps from the investor. The purpose of our fund is to streamline these actions — collecting, distributing, and reinvesting automatically. Our StarGate contract handles all of this autonomously. Every investor in our StarGate fund receives shares — a portion of the fund. The fund's profits are reflected in the increasing share price, making it a straightforward share-based system that is easily understood by the average investor.
Method	Manual Analysis, Functional Testing, Automated Testing
Blockchain	EVM
Audit Scope	The Scope of the Audit is to Analyse the Security ,Code Quality and Correctness of Cybro Vault Codebase
Contracts In Scope	https://github.com/cybro-io/vault-contracts.git
Fixed In	https://github.com/cybro-io/vault-contracts/pull/9
Project URL	https://cybro.io/en

Language	Solidity
Review 1	5th November 2024 - 29th November 2024
Updated Code Received	4th December 2024
Review 2	4th December 2024 - 5th December 2024

Number of Issues per Severity



High	5 (55.56%)
Medium	2 (22.22%)
Low	1 (11.11%)
Informational	1 (11.11%)

Severity

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

Checked Vulnerabilities

- ☑ Implicit visibility level
- ☑ Unsafe type inference
- ☑ Style guide violation
- ☑ Re-entrancy
- ☑ Timestamp Dependence
- ☑ Gas Limit and Loops
- ☑ Using inline assembly
- ☑ Using throw
- ☑ Upgradeable safety
- ☑ Using delegatecall
- ☑ Using suicide
- ☑ Multiple Sends
- ☑ Revert/require functions
- ☑ Private modifier
- ☑ Missing Zero Address Validation
- ☑ Return values of low-level calls
- ☑ Tautology or contradiction
- ☑ Dangerous strict equalities
- ☑ ERC's conformance
- ☑ Integer overflow/underflow
- ☑ Divide before multiply
- ☑ Address hardcoded
- ☑ Compiler version not fixed
- ☑ Malicious libraries
- ☑ Use of tx.origin
- ☑ Gasless Send
- ☑ Exception Disorder
- ☑ SWC Registry
- ☑ Race conditions/front running
- ☑ Arithmetic Computations Correctness
- ☑ Logical issues and flaws
- ☑ Improper or missing events

- [x] Ether theft
- [x] Arbitrary write to storage
- [x] Centralization of control
- [x] Access Management

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 Issues

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

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.

Issue Status

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

Evm panic in redeem function when msg.sender has allowance

Resolved

Path

vault-contracts/src/BaseVault.sol

Function

_applyPerformanceFee()

Description

The redeem() function may cause a VM panic during execution if the msg.sender has a balance of zero. This occurs in the _applyPerformanceFee function where the division is performed.

If balanceOf(msg.sender) is zero, the division triggers a VM panic, causing the entire redeem() function to fail. This is problematic when msg.sender is not the owner and does not hold shares themselves but has been granted an allowance to spend the owner's shares.

This can lead to Transaction Failures: The redeem() function will revert when msg.sender has zero balance but is authorized to redeem on behalf of the owner.

Recommendation

Modify _applyPerformanceFee to handle cases where the msg.sender is redeeming on behalf of another account. If balanceOf(msg.sender) is zero, skip the fee calculation for msg.sender to avoid panic.

Teams Comment

QuillAudits Team: The issue is not Fixed, If the owner has approved to other address to spend tokens, the EVM will throw the same error.

Precision loss and potential overflow in _deposit function due to multiplication after division

Partially Resolved

Path

vault-contracts/src/StargateVault.sol

Function

_deposit()

Description

The _deposit() function calculates assetToDeposit using the formula:

uint256 assetToDeposit = uint64(assets / _convertRate) * _convertRate;

This approach introduces two critical issues:

1. Precision Loss:

- When assets is not a perfect multiple of _convertRate, the division assets / _convertRate truncates the fractional part. This results in precision loss, and the calculated assetToDeposit will be less than the intended value.

- If the truncated value is zero (e.g., assets < _convertRate), the transaction will revert due to a subsequent operation requiring assetToDeposit > 0.

2. Overflow Risk: The multiplication uint64(assets / _convertRate) * _convertRate can cause an overflow if _convertRate or assets exceeds the uint64 limit after casting. This could result in incorrect calculations and unexpected behavior.

Recommendation

1. Avoid Precision Loss: Use SafeMath or equivalent operations to calculate the floor without truncating precision. Check for scenarios where assets < _convertRate and handle these cases explicitly.

2. Avoid Overflow Risk: Ensure all calculations use appropriate data types (uint256) without unnecessary downcasting. Validate inputs before performing operations.

Teams Comment

Cybro Team Comment: This precision loss is intended and negligible. It is done to avoid hitting this check in stargate vault
https://github.com/stargate-protocol/stargate-v2/blob/2a98da9b6ec2bb799f63bb264d1f331ee730df06/packages/stg-evm-v2/src/StargatePoolNative.sol#L75

This also does not affect user's funds because the remainder is always being refunded.
Even though this does not have any impact on users we've decided to update this logic to use SafeCast to be more future-proof

Precision loss in _getAmounts function due to integer division

Acknowledged

Path

vault-contracts/src/BlasterSwapV2Vault.sol

Function

_getAmounts()

Description

This implementation introduces a precision loss due to integer division. Specifically, when the amount is an odd number, the division amount / 2 truncates the decimal, causing the result to be slightly smaller than expected. This leads to an imbalance between amountFor0 and amountFor1, resulting in the total being less than the original amount by 1.

For example, if amount = 5:

amountFor0 = 5 / 2 = 2 (truncated).

amountFor1 = 5 - 2 = 3.

So, the total sum amountFor0 + amountFor1 equals 5, but this is only possible because amountFor1 is effectively one unit larger than the truncated result.

Recommendation

To avoid precision loss, ensure the amounts for amountFor0 and amountFor1 are split as evenly as possible by rounding up the division result when needed.

totalAssets() function might return wrong value because of hardcoded division

Acknowledged

Function

totalAssets()

Description

The totalAssets() function calculates the total assets by multiplying the pool's balance by the exchange rate and then dividing by 1e18. However, this division is hardcoded and may lead to an incorrect result if the exchangeRateStored() function doesn't return a value that properly scales to a 1e18 denominator.

The function assumes that the exchangeRateStored() value is always scaled by 1e18 (i.e., 18 decimals), but the description provided by Compound documentation suggests that the exchange rate is actually scaled by 1 * 10^(18 - 8 + Underlying Token Decimals) (i.e., accounting for 8 decimals of the underlying token). This mismatch in expected scaling could cause precision errors and incorrect results in the total assets calculation.

Teams Comment

QuillAudits Audit team: Better to use exchangeRateCurrent() function, instead of exchangeRate-Stored(), because the later one does not accrue interest before calculating the exchange rate.

Vault is vulnerable to first depositor inflation attack

Partially Resolved

Path

src/BaseVault.sol#L120

Function

deposit()

Description

Currently, when depositing to a vaultl, the way shares are calculated is the following:

1. If totalAssetsBefore is 0, mint shares equivalent to the asset

2. If totalAssetsBefore is not 0, mint sharess equivalent to `totalSupply() * increase / totalAssetsBefore`

It is a classic attack with the following flow:

1. The user is the first to interact with the vault, he sends a deposit transaction.

2. Attacker frontruns user's tx by performing 2 actions:

a) deposit 1 wei asset thus minting 1 wei share;

b) donate a big amount of asset to the vault.

3. User's transaction executes and the user mints 0 shares because of rounding in a formula that is used inside ERC4626: `shares = assets * sharesSupply / totalAssets`. That is because `sharesSupply` is 1 and `totalAssets` is large due to donation.

4. Then the attacker withdraws donated assets plus the user's deposit.

Recommendation

I like how this issue is handled in uniswapV2 so would suggest that or Implement the concept of virtual shares, similar to the ERC4626 OZ contract. More info about this concept here.

Medium Severity Issues

withdrawFunds function allows admin to steal funds from buffer and yield vaults

Acknowledged

Path

src/BaseVault.sol, src/BufferVault.sol & src/YieldStakingVault.sol

Function

withdrawFunds()

Description

The withdrawFunds() function allows an account with the DEFAULT_ADMIN_ROLE to withdraw any tokens or ETH held in the contract. Both the buffer vault and yield vault implementations override the _validateTokenToRecover() function to always return true, effectively making no distinction between legitimate and illegitimate tokens.

This setup provides unrestricted access to all funds held in the contracts, which could be misused by an admin with malicious intent or in the event of a compromised admin account. Consequently, the admin can withdraw funds from the vaults, potentially stealing user deposits or yield earnings.

Recommendation

Implement stricter controls on the withdrawFunds() function by introducing additional checks for legitimacy and purpose.

Legitimate tokens: Tokens not associated with user deposits or yield generation but sent accidentally to the contract (e.g., protocol-specific utility tokens).

Deadline should be passed as an argument and not block.timestamp

Acknowledged

Path

src/BlasterSwapV2Vault.sol

Function

BlasterSwapV2Vault::_swap()

Description

`block.timestamp` is used as the deadline for swaps in `BlasterSwapV2Vault::_swap()`.

In the PoS model, proposers know well in advance if they will propose one or consecutive blocks ahead of time. In such a scenario, a malicious validator can hold back the transaction and execute it at a more favourable block number e.g. when minAmountOut is no longer relevant.

This offers no protection as `block.timestamp` will have the value of whichever block the txn is inserted into, hence the txn can be held indefinitely by malicious validators.

Recommendation

Consider allowing function caller to specify swap deadline input parameter.

Teams Comment

Cybro team's Comment: Deadline should be passed as an argument and not block.timestamp this attach vector is considered unlikely as this contracts are mostly used on L2s

Low Severity Issues

depositedBalances mapping is not being subtracted in the redeem function

Acknowledged

Path

src/BaseVault.sol

Function

redeem()

Description

In the redeem function, the _depositedBalances[] mapping, which typically tracks the balances deposited by users, is not updated when shares are redeemed. This omission could lead to an inconsistency between the tracked deposited balances and the actual token balances.

As a result, the _depositedBalances[] mapping may still reflect the previous deposited value, even though shares have been redeemed and the associated assets have been withdrawn. This inconsistency could cause inaccurate calculations in subsequent operations, such as performance fee calculations, balance checks, or other accounting processes dependent on _depositedBalances[].

Recommendation

To ensure consistency, the _depositedBalances[] mapping must be decremented by the appropriate amount when shares are redeemed. The amount to subtract can be calculated using the shares being redeemed and the user's total deposited balance.

Teams Comment

QuillAudit team: Best if it can be subtracted in the redeem function itself instead of _applyPerformanceFee() function.

Informational Severity Issues

Use of slot0() and globalState() to get sqrtPrice-LimitX96 can lead to price manipulation.

Acknowledged

Path

src/BlasterSwapV3Vault.sol & src/AlgebraVault.sol

Function

getCurrentSqrtPrice()

Description

The function getCurrentSqrtPrice() uses UniswapV3.slot0 to get the value of sqrtPriceX96, which is used to perform the swap. However, the sqrtPriceX96 is pulled from Uniswap.slot0, which is the most recent data point and can be manipulated easily via MEV bots and Flashloans with sandwich attacks; which can cause the loss of funds when interacting with the Uniswap.swap function.

the function getCurrentSqrtPrice() uses AlgebraPool.globalState() to get the value of sqrtPriceX96, which is used to perform the swap. However, the sqrtPriceX96 is pulled from globalState(), which is the most recent data point and can be manipulated easily via MEV bots and Flashloans with sandwich attacks; which can cause the loss of funds when interacting with the AlgebraPool.swap function.

Recommendation

Use the TWAP function to get the value of sqrtPriceX96.

Functional Tests

Some of the tests performed are mentioned below:

- ✔ Should be able to mint tokens successfully to caller.
- ✘ Should be able to execute redeem() only when vault is not paused. (Intended Behaviour)
- ✔ Should be able to burn shares in redeem().
- ✔ Should be able to transfer ownership.
- ✔ Should be able to receive relative asset of shares burnt.
- ✔ Should be able to revert to zero amount deposit.
- ✔ Owner should not be able to withdraw pool tokens.

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 Vaults. 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

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Cybro Vaults. 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 Cybro Vaults. 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 sub-sequent to and despite using our audit services. It is the responsibility of Cybro Vaults to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

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

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

Follow Our Journey

AUDIT REPORT

December, 2024

For

Canada, India, Singapore, UAE, UK

www.quillaudits.com

mailto:audits@quillhash.com