HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Cinch Protocol
Date: 12 May, 2023

Document

Name	Smart Contract Code Review and Security Analysis Report for Cinch Protocol
Approved By	Marcin Ugarenko | Lead Solidity SC Auditor at Hacken OU
Type	Yield Farming;
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://www.cinchprotocol.io/
Changelog	19.04.2023 - Initial Review 09.05.2023 - Second Review 12.05.2023 - Third Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Cinch Protocol (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/cinchprotocol/contracts
Commit	ab6639e019f64ca27c22876c9b62a3ca2baedf73
Functional Requirements	https://docs.google.com/document/d/1PTq_WxOf-07qjiAOD10d6FHtDdB3GeEBK-iaBaloqv0/ https://docs.google.com/document/d/12nkopFwwz0xqZGNzzpnlJ63a--mSYwqaBId5bz2a2Do/
Technical Requirements	https://github.com/cinchprotocol/contracts/blob/main/v1/hardhat/docs/index.md
Contracts	File: ./v1/hardhat/contracts/revenueshare/GeneralRevenueShareLogic.sol SHA3: b59f843ca1d7f206a22c2e0eef250b70a7593dc21c1ed750af4dada157479b18 File: ./v1/hardhat/contracts/revenueshare/GeneralYieldSourceAdapter.sol SHA3: a9d3455976e3acf3935e9c1114831ce172e6d0c4e6368dcebd637c58e27a25f1 File: ./v1/hardhat/contracts/revenueshare/RevenueShareVault.sol SHA3: 6b0a01a05a96fa1ba73e17c209b729bb52d0db2412b710689ae5443bc95870ab File: ./v1/hardhat/contracts/revenueshare/RevenueShareVaultDHedge.sol SHA3: 13610f7ac82f8bc752a851d59bc66eb8e5f5d31261617282bc8e3ae0397ad79a File: ./v1/hardhat/contracts/revenueshare/RevenueShareVaultRibbonEarn.sol SHA3: bca0f9d8b9d30753f0118142c19b52e32bf41a05a78f06aa940e6d44f26b1efb File: ./v1/hardhat/contracts/revenueshare/interfaces/IYieldSourceContract.sol SHA3: 8b5417483fd0526d13805bb2e3d5dc74a9e816112653e7dccfdd40f5cfc53bc4 File: ./v1/hardhat/contracts/revenueshare/interfaces/IYieldSourceDHedge.sol SHA3: 2f125bc2a4d6c51cd75233c04b9c62f63ecef55cccd23924f64676401fe90fc4

Second review scope

Repository	https://github.com/cinchprotocol/contracts

Commit	3c7974006129490c308bb04856b98ad0c82fc522
Functional Requirements	https://docs.google.com/document/d/1PTq_WxOf-07qjiAOD10d6FHtDdB3GeEBK-iaBalogv0/ https://docs.google.com/document/d/12nkopFwwz0xqZGNzzpnlJ63a--mSYwqaBId5bz2a2Do/ https://docs.cinchprotocol.io/cinch-protocol-documentation/
Technical Requirements	https://github.com/cinchprotocol/contracts/blob/main/v1/hardhat/docs/index.md
Contracts	File: ./GeneralRevenueShareLogic.sol SHA3: a5fe161a257ea342ba840d4f9216e435e652db40d701966a65cf8814586ab56b File: ./GeneralYieldSourceAdapter.sol SHA3: 2eed6405702f9f5d058e1965a05cd32dea06264a60cdfc2a40d53e3b42611691 File: ./RevenueShareVault.sol SHA3: 1a1094f1cf804d8ba651d92db63454f42f8c3ff0b02e3dd94786d64c629d8778 File: ./RevenueShareVaultDHedge.sol SHA3: df45c34251280d1450c5cc364646727c9255edc634fc3081dc278fb634540e43 File: ./RevenueShareVaultRibbonEarn.sol SHA3: dc7d15ea6b4be81bb2f28c657a3e21b063886a4115850082c046f876dd820376 File: ./interfaces/IYieldSourceContract.sol SHA3: c6b8d8fe6f75f83c8abce7d1a656ece683fd110999b82d744f74079e203e9cf2 File: ./interfaces/IYieldSourceDHedge.sol SHA3: ed5e167c1ed0995bbd159125bebd92c67abce2f45ef7c6df6eee23575e3eb5b7 File: ./interfaces/IYieldSourceRibbonEarn.sol SHA3: 3601384088ac4090e6370910b9caf49f78ec6c8a8ca7e2f204660451f0a23c97 File: ./security/DepositPausableUpgradeable.sol SHA3: 60bc8be155fc79a3e620363fef9dc58bc96289ba61dd2b36ce1e2d2ce066ee76

Third review scope

Repository	https://github.com/cinchprotocol/contracts
Commit	91a2896f7f6675471326ba4b7e6d3700eb81c314
Functional Requirements	https://docs.google.com/document/d/1PTq_WxOf-07qjiAOD10d6FHtDdB3GeEBK-iaBalogv0/ https://docs.google.com/document/d/12nkopFwwz0xqZGNzzpnlJ63a--mSYwqaBId5bz2a2Do/ https://docs.cinchprotocol.io/cinch-protocol-documentation/
Technical Requirements	https://github.com/cinchprotocol/contracts/blob/main/v1/hardhat/docs/index.md
Contracts	File: ./GeneralRevenueShareLogic.sol SHA3: 8f9d85b364991f21d6a825ba4e872238f8eb1a94d0aec5e3b2a4490cbe604ec4 File: ./GeneralYieldSourceAdapter.sol SHA3: a5c0d045635ebdfb3f0dffc72bc567265b2f5ccf17116a52e83ade903d842f44 File: ./RevenueShareVault.sol

	SHA3: 1a1094f1cf804d8ba651d92db63454f42f8c3ff0b02e3dd94786d64c629d8778 File: ./RevenueShareVaultDHedge.sol SHA3: f60c4b8ef0b5ab56388554717064ca35e67694a2f8977b9c7863072bf6428a1d File: ./RevenueShareVaultRibbonEarn.sol SHA3: 2bee323d68cba419061a02595b94bd11909e4d4d08048eb4a59b02f88b0f8bf9 File: ./interfaces/IYieldSourceContract.sol SHA3: c6b8d8fe6f75f83c8abce7d1a656ece683fd110999b82d744f74079e203e9cf2 File: ./interfaces/IYieldSourceDHedge.sol SHA3: ed5e167c1ed0995bbd159125bebd92c67abce2f45ef7c6df6eee23575e3eb5b7 File: ./interfaces/IYieldSourceRibbonEarn.sol SHA3: 3601384088ac4090e6370910b9caf49f78ec6c8a8ca7e2f204660451f0a23c97 File: ./security/DepositPausableUpgradeable.sol SHA3: 60bc8be155fc79a3e620363fef9dc58bc96289ba61dd2b36ce1e2d2ce066ee76

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **8** out of **10**.

- NatSpec is detailed.
- Run instructions are provided.
- Project overview is detailed.
- Use cases are described.
- The technical documentation for integration implementation is partially provided.

Code quality

The total Code Quality score is **10** out of **10**.

- Development environment is configured.
- Code follows best practices.

Test coverage

Code coverage of the project is **100%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is present.
- Interactions by several users are tested.

Security score

As a result of the audit, the code contains no issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.8**.

1
2
3
4
5
6
7
8
9
10
The final score

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
19 April 2023	10	2	3	0

09 May 2023	6	0	1	0
12 May 2023	0	0	0	0

Risks

- Integration with Ribbon Finance is only done in a limited form, allowing users to forward deposits in their name. Withdrawal of funds is left to be managed by users outside of the Cinch Protocol. Integration is centralized and requires maintenance operations from the contract owner.
- Cinch protocol integrates external protocols into its system, which can introduce various risks and security concerns. The correctness of the integrations is out of the audit scope.

System Overview

Cinch Protocol - is referral system for blockchain applications aimed at democratizing economic opportunity by providing accessible decentralized financial products through seamless integration with wallets, with the following contracts:

- *GeneralRevenueShareLogic* - an abstract smart contract that implements revenue share business logic with a referral system.
- *GeneralYieldSourceAdapter* - an abstract smart contract that serves as a template for creating yield source adapters that interact with various yield source vaults in a revenue-sharing system. The contract provides a set of functions to interact with yield source vaults, such as depositing assets, redeeming assets based on shares, converting assets to shares, and converting shares to assets.
- *RevenueShareVault* - a smart contract inherited from GeneralRevenueShareLogic and *GeneralYieldSourceAdapter* enables users to deposit and withdraw assets into a yield source product, with support for referrals. It provides functionalities for managing asset deposits, share minting, asset redemptions, and querying account balances with pausable and access control features.
- *RevenueShareVaultDHedge - a* smart contract that extends *RevenueShareVault* and specializes in depositing and redeeming assets in dHedge yield source vaults. It provides conversion functions for assets to yield source shares, calculating share price, and querying account balances in the yield source.
- *RevenueShareVaultRibbonEarn* - a smart contract that extends *RevenueShareVault* and specializes in depositing assets into *Ribbon Earn* yield source vaults. It provides functions for depositing assets, querying share price and balances, and converting assets to and from yield source shares.
- *IYieldSourceContract* - an interface for a yield source contract that enables depositing and redeeming assets, as well as querying share price, total supply, and individual account balances.
- *IYieldSourceDHedge* - an interface that enables depositing funds, withdrawing funds, querying token price, total supply, and account balances.
- IYieldSourceDHedgeSwapper - an interface that enables withdrawing underlying assets while swapping them to a specified asset with slippage protection.

Privileged roles

- The owner of the *GeneralRevenueShareLogic* can manage referrals, adjust share allocations, and modify the Cinch performance fee

percentage, allowing them to control revenue share distribution and enable deposits and withdrawals for the revenue shares.
- The owner of the *GeneralYieldSourceAdapter* can update the address of the yield source vault.

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed

Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of Unused Variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP Standards Violation	EIP	EIP standards should not be violated.	Passed
Assets Integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances Manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed

Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply Manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style Guide Violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

H01. Invalid Calculations

In the *_convertAssetsToYieldSourceShares()* and *_convertYieldSourceSharesToAssets()* functions, calculations are done incorrectly.

The calculations in the *_convertAssetsToYieldSourceShares()* and *_convertYieldSourceSharesToAssets()* functions are highly dependent on the integrated protocol and are prone to error.

The implementations done in the GeneralYieldSourceAdapter contract are not taking into account the assets already stored in the integrated protocol or the total supply of the shares.

In the GeneralYieldSourceAdapter abstract contract, those functions should not contain a body and should be virtual without any implementation.

Implementation of those functions should only be done in top-level contracts where the protocol integration is done, this is the case in the *RevenueShareVaultRibbonEarn* and *RevenueShareVaultDHedge* contracts by overriding the implementation from abstract contract.

Unfortunately, the invalid logic is copied from the abstract contract to the final implementation.

As the Cinch Protocol Revenue Share Vaults are only an intermediary for the user funds to the integrated protocols, all the calculations for conversions between assets and shares should be implemented based on the integrated protocol conversions implementations.

The incorrect calculations in those functions affect the accuracy of the *mint()*, *withdraw()*, and *withdrawWithReferral()* functions initially. However, other functions are also affected.

The Cinch Protocol has been heavily tested, but it is using incorrect and limited mock implementations of integrated protocols. The assumptions in mock implementations that the share price is always *1 wei*, or that the conversion ratio from asset to share and from share to asset is always *1-to-1* are incorrect.

Paths:
./contracts/revenueshare/GeneralYieldSourceAdapter.sol:
_convertAssetsToYieldSourceShares(),
_convertYieldSourceSharesToAssets()
./contracts/revenueshare/RevenueShareVaultDHedge.sol:

_convertAssetsToYieldSourceShares(),
_convertYieldSourceSharesToAssets()
./contracts/revenueshare/RevenueShareVaultRibbonEarn.sol:
_convertAssetsToYieldSourceShares(),
_convertYieldSourceSharesToAssets()

Recommendation: Remove the implementation body of the *_convertAssetsToYieldSourceShares()* and *_convertYieldSourceSharesToAssets()* functions from the abstract contract; this will also require marking the RevenueShareVault as abstract.

Update the conversion calculations in the *RevenueShareVaultDHedge* and *RevenueShareVaultRibbonEarn* based on those protocols' implementations, directly extract the assets' value and total supply of shares from those protocols, and use them to correctly perform the conversion.

Update the mock protocol implementations to better reflect the real values. For example, the share price should be equal to 1289033757439016251 and the total shares supply should be 7454095482755680176243. Conversion ratios cannot be 1-to-1.

Consider other functions implemented in the GeneralYieldSourceAdapter that should be abstract, with implementation done in the specific yield source vault.

Found in: ab6639e

Status: Fixed (Revised commit: 3c79740)

H02. Data Consistency

In the *setYieldSourceVault()* function, the *yieldSourceVault* storage variable can be changed.

Changing this variable in already operating Revenue Share Vaults will lead to the user funds being locked, as the shares stored in the Revenue Share Vaults will not reflect the participation in the new Yield Source Vault.

Only the *yieldSourceSwapper* storage variable appears to be changeable, but there is no functionality for it.

Path:
./contracts/revenueshare/GeneralYieldSourceAdapter.sol:
setYieldSourceVault()

Recommendation: Remove the ability to change the *yieldSourceVault* variable for Revenue Share Vaults in which shares of the integrated protocol are stored in the contract.

Found in: ab6639e

Status: Fixed (Revised commit: 3c79740)

H03. Denial of Service; Ambiguous Third-Party Integration

In the *_redeemFromYieldSourceVault()* function in the RevenueShareVaultDHedge contract, the integration with DhedgeEasySwapper is done incorrectly.

The calculation of the *expectedAmountOut* value uses an incorrect *_convertYieldSourceSharesToAssets()* function, which returns the USD value of the shares with 18 decimal places.

The underneath asset of the Revenue Share Vault can be any asset, and as such, the *expectedAmountOut* will be incorrect if the asset is not a stablecoin.

Even if the underlying asset is a stablecoin such as USDC, the integration is potentially vulnerable to a Denial of Service.

Function expects that assets withdrawn from shares after all swaps done in the *DhedgeEasySwapper* contract will be worth more or equal to the *expectedAmountOut*; however, in most cases, this will not be true, as slippage needs to be accounted for in the calculation of the *expectedAmountOut*.

The real price of the underneath asset must also be taken into account in calculations, as even stablecoins can depeg, so the use of an Oracle will be beneficial.

Path:
./contracts/revenueshare/GeneralYieldSourceAdapter.sol

Recommendation: The *_convertYieldSourceSharesToAssets()* function needs to be updated to work correctly for any underlying asset and calculate the maximum expected amount out of any asset.

Minimal slippage should be accounted to the *expectedAmountOut*, for example, 1% to mitigate the potential DoS when redeeming.

Consider using Oracles in expectedAmountOut calculations.

Found in: ab6639e

Status: Fixed (Revised commit: 3c79740)

H04. Race Condition; Data Consistency

In the *depositToRevenueShare()* function, the *assetsFrom_* function parameter is not checked to see if it is equal to the *msg.sender*.

This leads to a situation where any approval given to the *RevenueShareVault* contract by any user can be exploited by a malicious actor by running the *depositToRevenueShare()* on their behalf.

For example, the user is giving approval to deposit funds to the RevenueShareVault contract, and before the user performs the deposit, a malicious actor will perform depositToRevenueShare() with assetsFrom_ set to the example user address.

Even though there is no financial gain for the malicious actor, the state of the contract data is corrupted and user funds are lost.

Path:
./contracts/revenueshare/GeneralRevenueShareLogic.sol : depositToRevenueShare()

Recommendation: Remove the *assetsFrom_* function parameter and use msg.sender as the 'from' for the *depositToRevenueShare()* function execution.

Found in: 3c79740

Status: Fixed (Revised commit: 91a2896)

Medium

M01. Gas Limit in Loops

In the *setTotalSharesInReferralAccordingToYieldSource()* function, there are two nested for loops that are not bounded; the first loop operates on the limited length of the *_referralSet* variable, which is controlled by the contract owner.

However, the second for loop is taken from an unlimited length of the *_userSetByReferral* variable, which will constantly increase during the contract lifetime, resulting in the Denial of Service of the *setTotalSharesInReferralAccordingToYieldSource()* function, as the Gas cost of the execution will be more than the block limit.

Path:
./contracts/revenueshare/RevenueShareVaultRibbonEarn.sol:
setTotalSharesInReferralAccordingToYieldSource()

Recommendation: Consider introducing a function that operates on a bounded size and a predictable Gas consumption.

Found in: ab6639e

Status: Fixed (Revised commit: 3c79740)

M02. Unchecked Approve

The function does not use SafeERC20 library for checking the result of ERC20 token approval. Tokens may not follow ERC20 standard and return false in case of approve failure or not returning any value at all.

Paths:
./contracts/revenueshare/GeneralYieldSourceAdapter.sol:
_depositToYieldSourceVault()
./contracts/revenueshare/RevenueShareVaultDHedge.sol:

_depositToYieldSourceVault(), _redeemFromYieldSourceVault()
./contracts/revenueshare/RevenueShareVaultRibbonEarn.sol:
_depositToYieldSourceVault()

Recommendation: Use SafeERC20 library to interact with tokens safely.

Found in: ab6639e

Status: Fixed (Revised commit: 3c79740)

Low

L01. Missing Zero Address Validation

Address parameters are being used without checking against the possibility of 0x0.

This can lead to unwanted external calls to 0x0.

Paths:
./contracts/revenueshare/RevenueShareVault.sol: initialize()
./contracts/revenueshare/GeneralYieldSourceAdapter.sol:
setYieldSourceVault()
./contracts/revenueshare/GeneralRevenueShareLogic.sol:
setTotalSharesByReferral()

Recommendation: Implement zero address checks.

Found in: ab6639e

Status: Fixed (Revised commit: 3c79740)

L02. Function State Mutability Can Be Changed to Pure

The function *redeemWithReferral()* does not read or modify the variables of the state and, due to that, can be declared pure.

This can lower Gas taxes.

Path:
./contracts/revenueshare/RevenueShareVaultRibbonEarn.sol:
redeemWithReferral()

Recommendation: Change function state mutability to pure.

Found in: ab6639e

Status: Fixed (Revised commit: 3c79740)

L03. Style Guide Violation - Order of Functions

The provided projects should follow the official guidelines. According to the Solidity Style Guide - Order of Functions section, contracts should follow this order: *constructor, receive function, fallback function, external, public, internal, private.* Functions should be grouped according to their visibility. Additionally, within a grouping, place the *view* and *pure* functions last.

Paths:
./contracts/revenueshare/GeneralRevenueShareLogic.sol
./contracts/revenueshare/GeneralYieldSourceAdapter.sol
./contracts/revenueshare/RevenueShareVault.sol
./contracts/revenueshare/DepositPausableUpgradeable
./contracts/revenueshare/RevenueShareVaultDHedge
./contracts/revenueshare/RevenueShareVaultRibbonEarn

Recommendation: Follow the official Solidity Guidelines for Order of Functions.

Found in: ab6639e

Status: Fixed (Revised commit: 3c79740)

L04. Style Guide Violation - Order of Layout

The provided projects should follow the official guidelines. According to the Solidity Style Guide - Order of Layout section, contracts should follow this order: *Type declarations, State variables, Events, Errors, Modifiers, Functions.* In contracts mentioned below, *Events* are declared before *State variables*.

Paths:
./contracts/revenueshare/GeneralRevenueShareLogic.sol
./contracts/revenueshare/GeneralYieldSourceAdapter.sol
./contracts/revenueshare/RevenueShareVault.sol
./contracts/revenueshare/DepositPausableUpgradeable

Recommendation: Follow the official Solidity Guidelines for Order of Layout.

Found in: ab6639e

Status: Fixed (Revised commit: 3c79740)

L05. Gas Optimization

In the *depositToRevenueShare()* function, the for loop is unbounded and operates on the length of the added referrals; this value is limited by the contract owner.

Inside the for loop, the *totalSharesInReferral* storage variable is read multiple times, which is Gas inefficient.

Path:
./contracts/revenueshare/GeneralRevenueShareLogic.sol:
depositToRevenueShare()

Recommendation: Consider assigning the *totalSharesInReferral* storage variable to a local memory variable.

Found in: ab6639e

Status: Fixed (Revised commit: 3c79740)

L06. Best Practice Violation: Checks-Effects-Interactions

Events should be emitted before interactions with external contracts.

In the *withdrawFromRevenueShare()* function, an event is emitted after calling *safeTransfer()*.

Path:
./contracts/revenueshare/GeneralRevenueShareLogic.sol:
withdrawFromRevenueShare()

Recommendation: Events should be emitted before the external calls.

Found in: ab6639e

Status: Fixed (Revised commit: 3c79740)

L07. Contradiction: Missing Validation

The *cinchPerformanceFeePercentage* storage variable is set in the init function, but it is not validated in the same way as in the *setCinchPerformanceFeePercentage()* function.

Missing validation can lead to the contract being configured incorrectly and potentially to underflow calculations in the *depositToRevenueShare()* function.

Path:
./contracts/revenueshare/GeneralRevenueShareLogic.sol:
__GeneralRevenueShareLogic_init_unchained()

Recommendation: Consider adding validation checks in the __GeneralRevenueShareLogic_init_unchained() function.

Found in: ab6639e

Status: Fixed (Revised commit: 3c79740)

L08. Missing Event for Critical Value Update

In the *__GeneralRevenueShareLogic_init_unchained()* function, the value of the *cinchPerformanceFeePercentage* variable is updated, but no event is emitted.

As a result, users cannot subscribe to the events and check what is happening within the project.

Path:
./contracts/revenueshare/GeneralRevenueShareLogic.sol:
__GeneralRevenueShareLogic_init_unchained()

Recommendation: Critical state changes should emit events for tracking things off-chain.

Found in: ab6639e

Status: Fixed (Revised commit: 3c79740)

L08. Missing Event for Critical Value Update

In the *__GeneralYieldSourceAdapter_init_unchained()* function, the value of the *yieldSourceVault* variable is updated, but no event is emitted.

As a result, users cannot subscribe to the events and check what is happening within the project.

Path:
./contracts/revenueshare/GeneralYieldSourceAdapter.sol:
__GeneralYieldSourceAdapter_init_unchained()

Recommendation: Critical state changes should emit events for tracking things off-chain.

Found in: ab6639e

Status: Fixed (Revised commit: 3c79740)

L09. Contradiction: Parameter Name

In the *_depositToYieldSourceVault()* function, the parameter *assets_* has confusing naming; the more relevant name should be *amount_*, which will be in line with the function's NatSpec.

Path:
./contracts/revenueshare/RevenueShareVaultDHedge.sol:
_depositToYieldSourceVault()

Recommendation: Consider updating the parameter name to a more suitable one.

Found in: ab6639e

Status: Fixed (Revised commit: 91a2896)

L10. Best Practice Violation: Interface Declaration

The IYieldSourceRibbonEarn interface is defined within the RevenueShareVaultRibbonEarn.sol file, which also contains the implementation for the RevenueShareVaultRibbonEarn contract.

As a best practice, it is recommended to declare interfaces in separate files and import them as needed in relevant locations.

Path:
./contracts/revenueshare/RevenueShareVaultRibbonEarn.sol:
IYieldSourceRibbonEarn

Recommendation: Move IYieldSourceRibbonEarn interface to separate file.

Found in: ab6639e

Status: Fixed (Revised commit: 3c79740)

L11. Use Of Hard-coded Values

The GeneralRevenueShareLogic abstract contract uses the hard-coded value of 10000 in the code.

It is best practice to never use hard-coded values but declare constants in their place.

Path:
./contracts/revenueshare/GeneralRevenueShareLogic.sol

Recommendation: Consider using a constant variable.

Found in: 3c79740

Status: Fixed (Revised commit: 91a2896)

L12. Redundant Code

After removing the setYieldSourceVault() function, the use of OwnableUpgradeable in the GeneralYieldSourceAdapter contract is unnecessary.

Path:
./contracts/revenueshare/GeneralYieldSourceAdapter.sol

Recommendation: Consider removing redundant code.

Found in: 3c79740

Status: Fixed (Revised commit: 91a2896)

L13. Commented Code

Within the RevenueShareVaultDHedge contract, there is a commented code for the redeemWithReferral() function.

Commented code decreases code readability.

Path:
./contracts/revenueshare/RevenueShareVaultDHedge.sol

Recommendation: Consider removing commented code.

Found in: 3c79740

Status: Fixed (Revised commit: 91a2896)

L14. Redundant Import

Unused imports should be removed from the contracts.

Unused imports are allowed in Solidity and do not pose a direct security issue. It is best practice to avoid them as they can decrease readability.

The usage of `IYieldSourceContract` is unnecessary for the `GeneralYieldSourceAdapter` contract.

Path:
./contracts/revenueshare/GeneralYieldSourceAdapter.sol

Recommendation: Remove the redundant import.

Found in: 3c79740

Status: Fixed (Revised commit: 91a2896)

L15. Redundant Code

In the RevenueShareVaultDHedge contract, there is redundant code:

The `using MathUpgradeable for uint256;` is redundant.

In the RevenueShareVaultRibbonEarn, there is redundant code:

The `using MathUpgradeable for uint256;` and `using EnumerableSetUpgradeable for EnumerableSetUpgradeable.AddressSet;` are redundant.

The `EnumerableSetUpgradeable` import in RevenueShareVaultRibbonEarn is redundant.

Paths:
./contracts/revenueshare/RevenueShareVaultDHedge.sol
./contracts/revenueshare/RevenueShareVaultRibbonEarn.sol

Recommendation: Consider removing redundant code.

Found in: 3c79740

Status: Fixed (Revised commit: 91a2896)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.