



Smart Contract Security Audit Report



Table Of Contents

1 Executive Summary	_____
2 Audit Methodology	_____
3 Project Overview	_____
3.1 Project Introduction	_____
3.2 Vulnerability Information	_____
4 Code Overview	_____
4.1 Contracts Description	_____
4.2 Visibility Description	_____
4.3 Vulnerability Summary	_____
5 Audit Result	_____
6 Statement	_____

1 Executive Summary

On 2026.07.06, the SlowMist security team received the RiftSwap Labs team's security audit application for curator2.0, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.

Level	Description
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit
		Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

This project is a modular on-chain asset management protocol based on Ethereum, aiming to provide institutional investors with customizable on-chain fund products. The project decouples the core vault logic, fee module, accounting system, policy routing, and access control through a layered architecture, supporting both synchronous and asynchronous vault modes, multiple fee strategies (fixed basis point fees, high-water mark performance fees, time-accumulated fees, etc.), and flexible investment strategy configurations. The core vault adopts an "atomic settlement" design, with the keeper role completing the settlement of asynchronous requests and directly paying out the output assets in a single transaction. The fee system is modularly pluggable based on a hook point mechanism, automatically calculating and collecting fees at key process points such as deposits, redemptions, and settlements. The project uses OpenZeppelin's UUPS proxy mode to support contract upgrades and is equipped with independent Accountant contracts to maintain NAV calculations, Escrow contracts for asset bucketing and custody, and PolicyRouter and Guard contracts for compliance policy control.

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Administrative Role Removal May Leave Roles Unassigned	Authority Control Vulnerability Audit	Suggestion	Fixed
N2	Lack of Target Role Holder Validation When Setting Role Administrators	Authority Control Vulnerability Audit	Suggestion	Fixed
N3	Redundant Conditional Logic in the _adminOf Function	Others	Suggestion	Fixed
N4	Missing Core Attribute Validation When Setting NAV Oracle	Design Logic Audit	Medium	Acknowledged

NO	Title	Category	Level	Status
N5	Lack of Symmetry Coverage Check for Adapter Position Sources	Design Logic Audit	Medium	Fixed
N6	Slippage Check Fails to Account for Deflationary Token Impacts	Design Logic Audit	Suggestion	Acknowledged
N7	Incorrect Withdrawal Queue Query Address Causes Omission in Unsettled Asset Calculation	Design Logic Audit	High	Fixed
N8	Missing Validation of Actual Deposited Shares for Asynchronous Redemption Requests	Design Logic Audit	Suggestion	Fixed
N9	Lack of Consistency Check Between Target Vault Underlying Assets and Share Tokens	Design Logic Audit	Medium	Fixed
N10	Risk of Collision Conflicts in Default Adapter ID Generation Logic	Others	Low	Fixed
N11	Redundant Iteration in Enabled Adapter Counting Logic	Others	Suggestion	Fixed
N12	Self-Managed Upgrade Permissions Introduce Critical Security Risks	Authority Control Vulnerability Audit	Suggestion	Fixed
N13	Risk of Excessive Total Fee Rate Due to Stacked Multiple Fee Modules	Design Logic Audit	Suggestion	Fixed

NO	Title	Category	Level	Status
N14	Forced Third-Party Reward Claim May Cause Permanent Fund Lockup	Design Logic Audit	Suggestion	Fixed
N15	Single Price Oracle Source Lacks Cross-Verification Mechanism	Others	Suggestion	Acknowledged
N16	Asset Lockup Risk Arising from Conflicts Between Product Lockup Period and Redemption Window	Design Logic Audit	Suggestion	Fixed
N17	Unbounded Policy Module Array Triggers Gas DoS on Hook Execution	Denial of Service Vulnerability	Suggestion	Fixed
N18	Subscription Whitelist Validates Payer Instead of Beneficiary	Design Logic Audit	Low	Fixed
N19	Retrospective Fee Charging Caused by Improper Initialization of Performance Fee High-Water Mark	Design Logic Audit	Medium	Fixed
N20	Vault Operations Freeze When Management Fee Liabilities Exceed Current Assets	Design Logic Audit	Medium	Fixed
N21	The Adapter Enable Toggle Does Not Cover Critical Paths	Design Logic Audit	Low	Acknowledged
N22	Improper Upper Bound Check for Withdrawal Exit Fee	Others	Suggestion	Acknowledged

NO	Title	Category	Level	Status
N23	No Upper Bound on Supported Asset List	Others	Suggestion	Fixed
N24	Excessive Privilege Risks	Authority Control Vulnerability Audit	Medium	Acknowledged
N25	External Dependencies Disclosure	Others	Information	Acknowledged

4 Code Overview

4.1 Contracts Description

Audit Version:

<https://github.com/r25it/curator2.0>

commit: 4d1884fd2a0eb4373be14e41236871444bb233e7

Fixed Version:

<https://github.com/r25it/curator2.0>

commit: d9c05ce36f701e2900149949762ec3c8b61e0603

(This does not include any newly added HyperLiquid (HyperAdapter + Topnod) related contracts.)

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

AccessManaged			
Function Name	Visibility	Mutability	Modifiers
__AccessManaged_init	Internal	Can Modify State	onlyInitializing
_authorizeUpgrade	Internal	-	onlyProxy
_hasRole	Internal	-	-

AsyncRequestVaultAdapter			
Function Name	Visibility	Mutability	Modifiers
depositRequestCount	External	-	-
redeemRequestCount	External	-	-
depositRequestIdAt	External	-	-
redeemRequestIdAt	External	-	-
isTrackedDepositRequest	External	-	-
isTrackedRedeemRequest	External	-	-
_recordDepositRequest	Internal	Can Modify State	-
_recordRedeemRequest	Internal	Can Modify State	-
_requireClaimPlanAmount	Internal	-	-
_pruneDepositRequestIfClosed	Internal	Can Modify State	-
_pruneRedeemRequestIfClosed	Internal	Can Modify State	-
_decodeAsyncClaimData	Internal	-	-
_asyncPositions	Internal	-	-
_asyncPositionCount	Internal	-	-

AsyncRequestVaultAdapter			
_requireRequestCapacity	Private	-	-
_removeDepositRequest	Private	Can Modify State	-
_removeRedeemRequest	Private	Can Modify State	-
_position	Private	-	-

BaseVaultAdapter			
Function Name	Visibility	Mutability	Modifiers
VAULT	Public	-	-
ASSET	Public	-	-
SHARE	Public	-	-
__BaseVaultAdapter_init	Internal	Can Modify State	onlyInitializing
__BaseVaultAdapter_init_unchained	Internal	Can Modify State	onlyInitializing
_requireDepositAssets	Internal	-	-
_requireRedeemAssets	Internal	-	-
_approveVault	Internal	Can Modify State	-
_clearVaultApproval	Internal	Can Modify State	-
_requestRef	Internal	-	-
_executeSyncDeposit	Internal	Can Modify State	-
_executeSyncRedeem	Internal	Can Modify State	-
_depositToVault	Internal	Can Modify State	-

BaseVaultAdapter			
_redeemFromVault	Internal	Can Modify State	-

BaseAdapter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
adapterId	External	-	-
positions	External	-	-
__BaseAdapter_init	Internal	Can Modify State	onlyInitializing
__BaseAdapter_init_unchained	Internal	Can Modify State	onlyInitializing
_validateBasePlan	Internal	-	-
_requireNonZeroAmount	Internal	-	-
_requireAdapterBalance	Internal	-	-
_requireMinAmountOut	Internal	-	-
_completedResult	Internal	-	-
_pendingResult	Internal	-	-

BaseProductModuleInstaller			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
moduleId	Public	-	-
_existingComponent	Internal	-	-

BaseProductModuleInstaller			
_directComponent	Internal	-	-
_fixedERC1967Component	Internal	Can Modify State	-
_uupsComponent	Internal	Can Modify State	-
_deployProxyOnly	Internal	Can Modify State	-
_latestImplementation	Internal	-	-
_productExistsAndAsset	Internal	-	-
_productAsset	Internal	-	-
_topnodRouter	Internal	-	-
_componentAddress	Internal	-	-
_factory	Internal	-	-
_component	Internal	-	-

BaseSyncVaultAdapter			
Function Name	Visibility	Mutability	Modifiers
requiresEscrowInput	External	-	-
execute	External	Can Modify State	onlyACModule

CoreVaultBase			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
convertToShares	External	-	-

CoreVaultBase			
convertToAssets	External	-	-
crystallizeFees	External	Can Modify State	onlyRole nonReentrant
totalAssets	Public	-	-
totalSupply	Public	-	-
__CoreVaultBase_init	Internal	Can Modify State	onlyInitializing
__CoreVaultBase_init_unchained	Internal	Can Modify State	onlyInitializing
_depositAssetIntoEscrow	Internal	Can Modify State	-
_convertToShares	Internal	-	-
_convertToAssets	Internal	-	-
_convertToShares	Internal	-	-
_convertToAssets	Internal	-	-
_runPolicyBefore	Internal	-	-
_runPolicyAfter	Internal	Can Modify State	-
_previewFee	Internal	-	-
_runFeeBefore	Internal	Can Modify State	-
_runFeeAfter	Internal	Can Modify State	-
_runBeforeHooks	Internal	Can Modify State	-
_runAfterHooks	Internal	Can Modify State	-
_accountingSnapshot	Internal	-	-

FeeModuleBase			
Function Name	Visibility	Mutability	Modifiers
__FeeModuleBase_init	Internal	Can Modify State	onlyInitializing
_requireFeeRouter	Internal	-	-

FixedBpsFeeBase			
Function Name	Visibility	Mutability	Modifiers
setConfig	External	Can Modify State	onlyRole
beforeAction	External	-	-
afterAction	External	-	-
previewFee	External	-	-
onFeeSettled	External	-	-
supportsPoint	External	-	-
supportsQuote	External	-	-
_quote	Internal	-	-
getConfig	External	-	-
_initialize	Internal	Can Modify State	onlyInitializing
_setConfig	Internal	Can Modify State	-
_validateFeeBps	Internal	-	-
_settlementToken	Internal	-	-
_feeAmount	Internal	-	-
_supportsPoint	Internal	-	-

FixedBpsFeeBase

--	--	--	--

NAVFeeModuleBase

Function Name	Visibility	Mutability	Modifiers
_accountingSnapshot	Internal	-	-
_accountingSnapshotAt	Internal	-	-
_applyPrior	Internal	-	-

TopnodRouter

Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
setAdapterAllowed	External	Can Modify State	onlyRole
batchDepositERC20FromRecipient	External	Can Modify State	-
depositERC20FromRecipient	External	Can Modify State	-
onBatchOrderSettled	External	Can Modify State	-
onOrderSettled	External	Can Modify State	-
__TopnodRouter_init	Internal	Can Modify State	onlyInitializing
__TopnodRouter_init_unchained	Internal	Can Modify State	onlyInitializing
_validateBatch	Internal	-	-
_newTraceId	Internal	Can Modify State	-
_deriveTraceId	Internal	-	-

TopnodRouter			
_openOrder	Internal	Can Modify State	-
_closeOrder	Internal	Can Modify State	-
_requireAllowedAdapter	Internal	-	-
_requireTopnodProxy	Internal	-	-
_pullBatch	Internal	Can Modify State	-
_approveBatch	Internal	Can Modify State	-
_clearBatchApprovals	Internal	Can Modify State	-
_forwardBatch	Internal	Can Modify State	-
_appearedBefore	Internal	-	-
_totalAmountFor	Internal	-	-

ERC4626Adapter			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
__ERC4626Adapter_init	Internal	Can Modify State	onlyInitializing
__ERC4626Adapter_init_unchained	Internal	Can Modify State	onlyInitializing
_depositToVault	Internal	Can Modify State	-
_redeemFromVault	Internal	Can Modify State	-

ERC7540Adapter			
Function Name	Visibility	Mutability	Modifiers

ERC7540Adapter			
initialize	External	Can Modify State	initializer
execute	External	Can Modify State	onlyACModule
requiresEscrowInput	External	-	-
positions	External	-	-
__ERC7540Adapter_init	Internal	Can Modify State	onlyInitializing
__ERC7540Adapter_init_unchained	Internal	Can Modify State	onlyInitializing
_requestDeposit	Internal	Can Modify State	-
_requestRedeem	Internal	Can Modify State	-
_claimDeposit	Internal	Can Modify State	-
_claimRedeem	Internal	Can Modify State	-
_depositToVault	Internal	Can Modify State	-
_redeemFromVault	Internal	Can Modify State	-

ERC7575Adapter			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
__ERC7575Adapter_init	Internal	Can Modify State	onlyInitializing
__ERC7575Adapter_init_unchained	Internal	Can Modify State	onlyInitializing
_depositToVault	Internal	Can Modify State	-
_redeemFromVault	Internal	Can Modify State	-

HybridSyncDepositAsyncRedeemAdapter			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
execute	External	Can Modify State	onlyACModule
requiresEscrowInput	External	-	-
positions	External	-	-
__HybridSyncDepositAsyncRedeemAdapter_init	Internal	Can Modify State	onlyInitializing
__HybridSyncDepositAsyncRedeemAdapter_init_unchained	Internal	Can Modify State	onlyInitializing
_requestRedeem	Internal	Can Modify State	-
_claimRedeem	Internal	Can Modify State	-
_depositToVault	Internal	Can Modify State	-
_redeemFromVault	Internal	Can Modify State	-

JtrsAdapter			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
execute	External	Can Modify State	onlyACModule
requiresEscrowInput	External	-	-
positions	External	-	-
__JtrsAdapter_init	Internal	Can Modify State	onlyInitializing

JtrsyAdapter			
__JtrsyAdapter_init_unchained	Internal	Can Modify State	onlyInitializing
_requestDeposit	Internal	Can Modify State	-
_requestRedeem	Internal	Can Modify State	-
_claimDeposit	Internal	Can Modify State	-
_claimRedeem	Internal	Can Modify State	-
_depositToVault	Internal	Can Modify State	-
_redeemFromVault	Internal	Can Modify State	-

TopnodAdapter			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
TOPNOD_ROUTER	External	-	-
pendingOrderCount	External	-	-
pendingTraceIdAt	External	-	-
pendingOrderOf	External	-	-
execute	External	Can Modify State	onlyACModule
requiresEscrowInput	External	-	-
positions	External	-	-
onOrderSettled	External	Can Modify State	onlyTopnodRouter
onBatchOrderSettled	External	-	onlyTopnodRouter
__TopnodAdapter_init	Internal	Can Modify State	onlyInitializing

TopnodAdapter			
__TopnodAdapter_init_unchained	Internal	Can Modify State	onlyInitializing
_validatePlan	Internal	-	-
_decodeOrderData	Internal	-	-
_recordPendingOrder	Internal	Can Modify State	-
_requirePendingOrder	Internal	-	-
_removePendingOrder	Internal	Can Modify State	-
_addressToBytes32	Internal	-	-

VRPCQAdapter			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
execute	External	Can Modify State	onlyACModule
requiresEscrowInput	External	-	-
positions	External	-	-
__VRPCQAdapter_init	Internal	Can Modify State	onlyInitializing
__VRPCQAdapter_init_unchained	Internal	Can Modify State	onlyInitializing
_requestRedeem	Internal	Can Modify State	-
_claimRedeem	Internal	Can Modify State	-
_depositToVault	Internal	Can Modify State	-
_redeemFromVault	Internal	-	-

VRPCQAdapter			
_pendingWithdrawalAssets	Internal	-	-

AssetSummedNAVOracle			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
registerPositionAsset	External	Can Modify State	onlyRole
registerPositionSource	External	Can Modify State	onlyRole
positionAssets	External	-	-
positionSources	External	-	-
totalAssets	External	-	-
__AssetSummedNAVOracle_init	Internal	Can Modify State	onlyInitializing
__AssetSummedNAVOracle_init_unchained	Internal	Can Modify State	onlyInitializing
_registerPositionAsset	Internal	Can Modify State	-
_registerPositionSource	Internal	Can Modify State	-
_validateRegistryAssetCoverage	Internal	-	-
_valueInBaseAsset	Internal	-	-
_oracleValue	Internal	-	-

CoreVaultAsync			
Function Name	Visibility	Mutability	Modifiers

CoreVaultAsync			
initialize	External	Can Modify State	initializer
initialized	External	-	-
supportsInterface	External	-	-
setOperator	External	Can Modify State	-
pendingDepositRequest	External	-	-
claimableDepositRequest	External	-	-
pendingRedeemRequest	External	-	-
claimableRedeemRequest	External	-	-
maxDeposit	External	-	-
maxMint	External	-	-
maxWithdraw	External	-	-
maxRedeem	External	-	-
requestDeposit	External	Can Modify State	nonReentrant
requestDeposit	External	Can Modify State	nonReentrant
requestRedeem	External	Can Modify State	nonReentrant
previewDeposit	External	-	-
previewMint	External	-	-
previewWithdraw	External	-	-
previewRedeem	External	-	-
deposit	External	Can Modify State	nonReentrant

CoreVaultAsync			
deposit	External	Can Modify State	nonReentrant
mint	External	Can Modify State	nonReentrant
mint	External	Can Modify State	nonReentrant
redeem	External	Can Modify State	nonReentrant
withdraw	External	Can Modify State	nonReentrant
cancelRequest	External	Can Modify State	nonReentrant
fulfillDeposit	External	Can Modify State	onlyRole nonReentrant
fulfillRedeem	External	Can Modify State	onlyRole nonReentrant
__CoreVaultAsync_init	Internal	Can Modify State	onlyInitializing
__CoreVaultAsync_init_unchained	Internal	Can Modify State	onlyInitializing
_isCallerOrOperator	Internal	-	-
_requireControllerOrOperator	Internal	-	-
_validateRequestInputs	Internal	-	-
_requestDeposit	Internal	Can Modify State	-
_requestRedeem	Internal	Can Modify State	-
_validateClaim	Internal	-	-
_claim	Internal	Can Modify State	-
_pendingRequest	Internal	-	-

RequestManager			
Function Name	Visibility	Mutability	Modifiers

RequestManager			
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
requests	External	-	-
getRequest	External	-	-
pendingDepositRequest	External	-	-
claimableDepositRequest	External	-	-
pendingRedeemRequest	External	-	-
claimableRedeemRequest	External	-	-
createDepositRequest	External	Can Modify State	onlyCoreVault
createRedeemRequest	External	Can Modify State	onlyCoreVault
fulfillDeposit	External	Can Modify State	onlyCoreVault
fulfillRedeem	External	Can Modify State	onlyCoreVault
claimDepositByAssets	External	Can Modify State	onlyCoreVault
claimDepositByShares	External	Can Modify State	onlyCoreVault
claimRedeemByShares	External	Can Modify State	onlyCoreVault
claimRedeemByAssets	External	Can Modify State	onlyCoreVault
cancel	External	Can Modify State	onlyCoreVault
__RequestManager_init	Internal	Can Modify State	onlyInitializing
__RequestManager_init_unchained	Internal	Can Modify State	onlyInitializing
_claim	Internal	Can Modify State	-

RequestManager			
_claimableTotal	Internal	-	-
_updateAggregates	Internal	Can Modify State	-
_requireRequest	Internal	-	-
_requireController	Internal	-	-

CoreVaultSync			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
initialized	External	-	-
supportsInterface	External	-	-
deposit	External	Can Modify State	nonReentrant
deposit	External	Can Modify State	nonReentrant
mint	External	Can Modify State	nonReentrant
mint	External	Can Modify State	nonReentrant
withdraw	External	Can Modify State	nonReentrant
redeem	External	Can Modify State	nonReentrant
maxDeposit	External	-	-
maxMint	External	-	-
maxWithdraw	External	-	-
maxRedeem	External	-	-
previewDeposit	Public	-	-

CoreVaultSync			
previewMint	Public	-	-
previewWithdraw	Public	-	-
previewRedeem	Public	-	-
__CoreVaultSync_init	Internal	Can Modify State	onlyInitializing
__CoreVaultSync_init_unchained	Internal	Can Modify State	onlyInitializing
_assetsToShares	Internal	-	-
_sharesToAssets	Internal	-	-
_deposit	Internal	Can Modify State	-
_mint	Internal	Can Modify State	-
_withdraw	Internal	Can Modify State	-
_redeem	Internal	Can Modify State	-
_spendShareAllowance	Internal	Can Modify State	-
_tryConvertToAssetsRealtime	Internal	-	-
_previewConvertToSharesAfterFees	Internal	-	-
_previewConvertToAssetsAfterFees	Internal	-	-
_previewFeePricing	Internal	-	-
_tryApplyManagementFeePreview	Internal	-	-

Accountant			
Function Name	Visibility	Mutability	Modifiers

Accountant			
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
initialized	External	-	-
convertToShares	External	-	-
convertToAssets	External	-	-
setNAVOracle	External	Can Modify State	onlyRole
totalAssets	Public	-	-
totalAssets	Public	-	-
navPerShare	Public	-	-
navPerShare	Public	-	-
accountingSnapshot	Public	-	-
__Accountant_init	Internal	Can Modify State	onlyInitializing
__Accountant_init_unchained	Internal	Can Modify State	onlyInitializing
_navResult	Internal	-	-

ACModule			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
initialized	External	-	-
executeTrades	External	Can Modify State	onlyRole nonReentrant

ACModule			
ensureLiquidity	External	Can Modify State	onlyRole nonReentrant
executeTrade	Public	Can Modify State	onlyRole nonReentrant
settleAsyncTradeOutput	External	Can Modify State	-
__ACModule_init	Internal	Can Modify State	onlyInitializing
__ACModule_init_unchained	Internal	Can Modify State	onlyInitializing
_executeTrade	Internal	Can Modify State	-
_requireAdapterFor	Internal	-	-
_runPolicyBefore	Internal	-	-
_runPolicyAfter	Internal	Can Modify State	-

AdapterRegistry			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
registerAdapter	Public	Can Modify State	onlyRole
setAdapterEnabled	External	Can Modify State	onlyRole
setSupportedAsset	External	Can Modify State	onlyRole
requireAdapter	External	-	-
getAdapterInfo	External	-	-
requireAdapterFor	External	-	-

AdapterRegistry			
adapterOf	External	-	-
adapterIds	External	-	-
adapterCount	External	-	-
supportedAssets	External	-	-
__AdapterRegistry_init	Internal	Can Modify State	onlyInitializing
__AdapterRegistry_init_unchained	Internal	Can Modify State	onlyInitializing
_requireAdapterInfo	Internal	-	-
_setSupportedAsset	Internal	Can Modify State	-
_isAssetSupportedByAnyAdapter	Internal	-	-
_removeSupportedAsset	Internal	Can Modify State	-

Escrow			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
initialized	External	-	-
depositAsset	External	Can Modify State	onlyCoreVault
releaseAsset	External	Can Modify State	onlyAssetReleaser
mintShares	External	Can Modify State	onlyCoreVaultOrFeeRouter
burnShares	External	Can Modify State	onlyCoreVault
spendShareAllowance	External	Can Modify State	onlyCoreVault

Escrow			
transferShares	External	Can Modify State	onlyFeeRouter
depositShares	External	Can Modify State	onlyCoreVault
mintSharesToBucket	External	Can Modify State	onlyCoreVault
burnSharesFromBucket	External	Can Modify State	onlyCoreVault
moveBucket	External	Can Modify State	onlyCoreVault
receiveTradeOutput	External	Can Modify State	onlyTradeCaller
clearUntrackedAsset	External	Can Modify State	onlyRole
activeAssets	External	-	-
__Escrow_init	Internal	Can Modify State	onlyInitializing
__Escrow_init_unchained	Internal	Can Modify State	onlyInitializing
_validateTokenMove	Internal	-	-
_decreaseBucket	Internal	Can Modify State	-
_balanceOf	Internal	-	-

FeeRouter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
addFeeModule	External	Can Modify State	onlyRole
addFeeModuleAt	External	Can Modify State	onlyRole

FeeRouter			
removeFeeModule	External	Can Modify State	onlyRole
setTypedFeeModule	External	Can Modify State	onlyRole
beforeAction	External	Can Modify State	-
afterAction	External	Can Modify State	-
previewFee	External	-	-
previewMaxWithdrawShares	External	-	-
getFeeModules	External	-	-
getTypedFeeModule	External	-	-
totalManagementFees	External	-	-
totalHWMPPerformanceFees	External	-	-
__FeeRouter_init	Internal	Can Modify State	onlyInitializing
__FeeRouter_init_unchained	Internal	Can Modify State	onlyInitializing
_addFeeModule	Internal	Can Modify State	-
_requireVaultCaller	Internal	-	-
_runBeforeActionModules	Internal	Can Modify State	-
_runAfterActionModules	Internal	Can Modify State	-
_previewFee	Internal	-	-
_sameModules	Internal	-	-
_requireManagementFeeHooksMounted	Internal	-	-
_settleModuleResult	Internal	Can Modify State	-

FeeRouter			
_classifyModuleResult	Internal	-	-
_addDelta	Internal	-	-
_vaultTokens	Internal	-	-
_contains	Internal	-	-
_isSettlementPoint	Internal	-	-
_requireTransferSettlement	Internal	-	-

ShareToken			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
initialized	External	-	-
mint	External	Can Modify State	onlyMinter
burn	External	Can Modify State	onlyMinter
spendAllowance	External	Can Modify State	onlyMinter
moveShares	External	Can Modify State	onlyMinter
vault	External	-	-
supportsInterface	External	-	-
decimals	Public	-	-
__ShareToken_init	Internal	Can Modify State	onlyInitializing
__ShareToken_init_unchained	Internal	Can Modify State	onlyInitializing

ShareToken			

VaultPolicyRouter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
initialized	External	-	-
mountPolicy	External	Can Modify State	onlyRole
unmountPolicy	External	Can Modify State	onlyRole
setPolicyEnabled	External	Can Modify State	onlyRole
getPolicyModules	External	-	-
isMounted	External	-	-
isEnabled	External	-	-
beforeAction	External	-	-
afterAction	External	Can Modify State	-
__VaultPolicyRouter_init	Internal	Can Modify State	onlyInitializing
__VaultPolicyRouter_init_unchained	Internal	Can Modify State	onlyInitializing
_requireModuleSupportsPoint	Internal	-	-
_storeForSubject	Internal	-	-
_storeForSubjectView	Internal	-	-
_requireDispatchCaller	Internal	-	-

VaultPolicyRouter			
_isACModuleForVault	Internal	-	-
_dispatchBefore	Internal	-	-
_dispatchAfter	Internal	Can Modify State	-

AdapterModuleInstaller			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BaseProductModuleInstaller
install	External	Can Modify State	-
_installTopnodAdapter	Internal	Can Modify State	-
_validateSupportedAsset	Internal	-	-
_deployConfiguredVaultAdapter	Internal	Can Modify State	-
_deployERC4626	Internal	Can Modify State	-
_deployVaultAdapter	Internal	Can Modify State	-
_deploySharedTopnodAdapter	Internal	Can Modify State	-
_adapterImplementation	Internal	-	-
_requireRouterOperatorRole	Internal	-	-
_enabledAdapterCount	Internal	-	-
_adapterId	Internal	-	-
_anyEnabled	Internal	-	-
_sharedSupportedAssets	Internal	-	-
_topnodAdapterId	Internal	-	-

AdapterModuleInstaller			
_isSharedAdapterType	Internal	-	-
_kindReportsPositions	Internal	-	-
_adapterAssets	Internal	-	-
_adapterMetadata	Internal	-	-

FeeModuleInstaller			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BaseProductModuleInstaller
install	External	Can Modify State	-
_enabledFeeCount	Internal	-	-
_deployUUPSFixedBpsFee	Internal	Can Modify State	-
_installManagementFeeHooks	Internal	Can Modify State	-
_feeImplementation	Internal	-	-
_feeRecipient	Internal	-	-

GuardModuleInstaller			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BaseProductModuleInstaller
install	External	Can Modify State	-
_guardImplementation	Internal	-	-
_configureGuards	Internal	Can Modify State	-

GuardModuleInstaller			
_enabledGuardCount	Internal	-	-
_mountPolicy	Internal	Can Modify State	-

LifecycleModuleInstaller			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BaseProductModuleInstaller
install	External	Can Modify State	-
_enabledRuleCount	Internal	-	-
_mountPolicy	Internal	Can Modify State	-
_lockupStart	Internal	-	-
_lifecycleImplementation	Internal	-	-

NAVModuleInstaller			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BaseProductModuleInstaller
install	External	Can Modify State	-
_positionAssetInputs	Internal	-	-
_positionSourceIds	Internal	-	-
_navImplementation	Internal	-	-

RequiredCoreInstaller			
Function Name	Visibility	Mutability	Modifiers

RequiredCoreInstaller			
<Constructor>	Public	Can Modify State	BaseProductModuleInstaller
install	External	Can Modify State	-
_requiredImplementation	Internal	-	-
_vaultImplementationComponentId	Internal	-	-
_grantProductRoles	Internal	Can Modify State	-
_productRoles	Internal	-	-

ProductFactory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
registerInstaller	External	Can Modify State	onlyRole
registerImplementation	External	Can Modify State	onlyRole
registerImplementations	External	Can Modify State	onlyRole
createProduct	External	Can Modify State	onlyRole
installModules	External	Can Modify State	onlyRole nonReentrant
finalizeProduct	External	Can Modify State	onlyRole
productInfo	External	-	-
implementationOf	External	-	-
latestImplementationOf	External	-	-

ProductFactory			
installedModuleCount	External	-	-
isInstanceInstalled	External	-	-
componentCount	External	-	-
componentAt	External	-	-
componentOf	External	-	-
__ProductFactory_init	Internal	Can Modify State	onlyInitializing
__ProductFactory_init_unchained	Internal	Can Modify State	onlyInitializing
_registerImplementation	Internal	Can Modify State	-
_validateCreateConfig	Internal	-	-
_toProductInfo	Internal	-	-
_toVaultAuthorities	Internal	-	-
_configuredOrAdmin	Internal	-	-
_vaultAuthoritiesOrAdmin	Internal	-	-
_componentAddress	Internal	-	-
_validateModuleCall	Internal	-	-
_installModule	Internal	Can Modify State	-
_validateModuleOrder	Internal	-	-
_recordComponent	Internal	Can Modify State	-
_validateComponent	Internal	-	-
_delegateInstall	Internal	Can Modify State	-

ProductFactory			
_stageMaskForModule	Internal	-	-
_requireRequiredCoreComponents	Internal	-	-
_requireComponent	Internal	-	-
_isAsyncVault	Internal	-	-
_supportsInterface	Internal	-	-
_handoffProductAccess	Internal	Can Modify State	-
_productRoles	Internal	-	-

UninitializedERC1967Proxy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Payable	ERC1967Proxy
_unsafeAllowUninitialized	Internal	-	-

EntryFixedBpsFee			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
_settlementToken	Internal	-	-
_feeAmount	Internal	-	-
_supportsPoint	Internal	-	-
_feeOnGrossAssets	Internal	-	-

ExitFixedBpsFee			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
_settlementToken	Internal	-	-
_feeAmount	Internal	-	-
_supportsPoint	Internal	-	-
_withdrawNetShares	Internal	-	-
_validateFeeBps	Internal	-	-

HWMPerformanceFee			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
setPerformanceConfig	External	Can Modify State	onlyRole
beforeAction	External	-	-
afterAction	External	-	-
previewFee	External	-	-
onFeeSettled	External	Can Modify State	-
supportsPoint	External	-	-
supportsQuote	External	-	-
getPerformanceConfig	External	-	-

HWMPerformanceFee			
getPerformanceState	External	-	-
totalFees	External	-	-
previewCrystallize	External	-	-
previewCrystallizeAt	External	-	-
__HWMPerformanceFee_init	Internal	Can Modify State	onlyInitializing
__HWMPerformanceFee_init_unchained	Internal	Can Modify State	onlyInitializing
_setPerformanceConfig	Internal	Can Modify State	-
_effectiveHighWaterMark	Internal	-	-
_currentNavPerShare	Internal	-	-
_previewPerformance	Internal	-	-
_performanceFeeAssets	Internal	-	-

TimeAccrualFee			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
setTimeAccrualConfig	External	Can Modify State	onlyRole
beforeAction	External	Can Modify State	-
afterAction	External	Can Modify State	-
previewFee	External	-	-

TimeAccrualFee			
onFeeSettled	External	Can Modify State	-
supportsPoint	External	-	-
supportsQuote	External	-	-
getTimeAccrualConfig	External	-	-
getTimeAccrualState	External	-	-
totalFees	External	-	-
previewCrystallize	External	-	-
previewCrystallizeAt	External	-	-
previewLiabilityAssets	External	-	-
__TimeAccrualFee_init	Internal	Can Modify State	onlyInitializing
__TimeAccrualFee_init_unchained	Internal	Can Modify State	onlyInitializing
_setTimeAccrualConfig	Internal	Can Modify State	-
_accrue	Internal	Can Modify State	-
_settleElapsed	Internal	Can Modify State	-
_settleElapsedAtCurrentRate	Internal	Can Modify State	-
_previewCrystallize	Internal	-	-
_pendingCrystallizableAssets	Internal	-	-
_snapshot	Internal	Can Modify State	-
_syncBasis	Internal	Can Modify State	-
_isBeforeAccrualPoint	Internal	-	-

TimeAccrualFee			
_isAfterSyncPoint	Internal	-	-

SubscriptionWhitelistGuard			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
moduleId	External	-	-
policyPoints	External	-	-
isWhitelisted	External	-	-
setWhitelisted	External	Can Modify State	onlyRole
setWhitelistedBatch	External	Can Modify State	onlyRole
beforeAction	External	-	-
afterAction	External	-	-
__SubscriptionWhitelistGuard_init	Internal	Can Modify State	onlyInitializing
__SubscriptionWhitelistGuard_init_unchained	Internal	Can Modify State	onlyInitializing
_setWhitelisted	Internal	Can Modify State	-
_requireWhitelisted	Internal	-	-
_requirePolicyRouter	Internal	-	-

VaultLimitGuard			
Function Name	Visibility	Mutability	Modifiers

VaultLimitGuard			
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
moduleId	External	-	-
policyPoints	External	-	-
minDeposit	External	-	-
maxDeposit	External	-	-
minRedeem	External	-	-
maxRedeem	External	-	-
minRedeemAssets	External	-	-
maxTvlCap	External	-	-
maxCap	External	-	-
setMinDeposit	External	Can Modify State	onlyRole
setMaxDeposit	External	Can Modify State	onlyRole
setMinRedeem	External	Can Modify State	onlyRole
setMaxRedeem	External	Can Modify State	onlyRole
setMinRedeemAssets	External	Can Modify State	onlyRole
setMaxTvlCap	External	Can Modify State	onlyRole
setMaxCap	External	Can Modify State	onlyRole
beforeAction	External	-	-
afterAction	External	-	-

VaultLimitGuard			
__VaultLimitGuard_init	Internal	Can Modify State	onlyInitializing
__VaultLimitGuard_init_unchained	Internal	Can Modify State	onlyInitializing
_setInitialLimits	Internal	Can Modify State	-
_setMinDeposit	Internal	Can Modify State	-
_setMaxDeposit	Internal	Can Modify State	-
_setMinRedeem	Internal	Can Modify State	-
_setMaxRedeem	Internal	Can Modify State	-
_setMinRedeemAssets	Internal	Can Modify State	-
_setMaxTvlCap	Internal	Can Modify State	-
_setMaxCap	Internal	Can Modify State	-
_checkDepositLimits	Internal	-	-
_checkRedeemShareLimits	Internal	-	-
_checkMaxTvlCap	Internal	-	-
_pendingDepositAssets	Internal	-	-
_bucketBalance	Internal	-	-
_checkMaxCap	Internal	-	-
_checkActualMaxCap	Internal	-	-
_assetsToShares	Internal	-	-
_sharesToAssets	Internal	-	-
_request	Internal	-	-

VaultLimitGuard			
_requirePolicyRouter	Internal	-	-

ProductLockRule			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
moduleId	External	-	-
policyPoints	External	-	-
getProductLockConfig	External	-	-
setProductLockConfig	External	Can Modify State	onlyRole
beforeAction	External	-	-
afterAction	External	-	-
__ProductLockRule_init	Internal	Can Modify State	onlyInitializing
__ProductLockRule_init_unchained	Internal	Can Modify State	onlyInitializing
_setProductLockConfig	Internal	Can Modify State	-
_requirePolicyRouter	Internal	-	-

RedemptionRule			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer

RedemptionRule			
moduleId	External	-	-
policyPoints	External	-	-
getRedemptionConfig	External	-	-
setRedemptionConfig	External	Can Modify State	onlyRole
beforeAction	External	-	-
afterAction	External	-	-
__RedemptionRule_init	Internal	Can Modify State	onlyInitializing
__RedemptionRule_init_unchained	Internal	Can Modify State	onlyInitializing
_setRedemptionConfig	Internal	Can Modify State	-
_requirePolicyRouter	Internal	-	-

SubscriptionRule			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
moduleId	External	-	-
policyPoints	External	-	-
getSubscriptionConfig	External	-	-
setSubscriptionConfig	External	Can Modify State	onlyRole
beforeAction	External	-	-
afterAction	External	-	-

SubscriptionRule			
__SubscriptionRule_init	Internal	Can Modify State	onlyInitializing
__SubscriptionRule_init_unchained	Internal	Can Modify State	onlyInitializing
_setSubscriptionConfig	Internal	Can Modify State	-
_requirePolicyRouter	Internal	-	-

OffchainAssetPriceOracle			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
getPrice	External	-	-
priceData	External	-	-
batchUpdatePrice	External	Can Modify State	onlyRole
setMaxPriceAge	External	Can Modify State	onlyRole
setDefaultMaxPriceAge	External	Can Modify State	onlyRole
updatePrice	Public	Can Modify State	onlyRole
__OffchainAssetPriceOracle_init	Internal	Can Modify State	onlyInitializing
__OffchainAssetPriceOracle_init_unchained	Internal	Can Modify State	onlyInitializing

AccessManager			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-

AccessManager			
initialize	External	Can Modify State	initializer
requireRole	External	-	-
setRoleAdmin	External	Can Modify State	onlyRole
grantRole	External	Can Modify State	onlyRole
batchGrantRoles	External	Can Modify State	-
batchSetRoleAdmin	External	Can Modify State	-
revokeRole	External	Can Modify State	onlyRole
renounceRole	External	Can Modify State	-
__AccessManager_init	Internal	Can Modify State	onlyInitializing
__AccessManager_init_unchained	Internal	Can Modify State	onlyInitializing
_checkRole	Internal	-	-
_grantRole	Internal	Can Modify State	-
_adminOf	Internal	-	-
_authorizeUpgrade	Internal	-	onlyProxy

ProjectFeeTreasury			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
initialize	External	Can Modify State	initializer
claim	External	Can Modify State	nonReentrant

ProjectFeeTreasury			
claimVerified	External	Can Modify State	nonReentrant
verify	External	Can Modify State	nonReentrant
setMerkleRoot	External	Can Modify State	onlyRole
__ProjectFeeTreasury_init	Internal	Can Modify State	onlyInitializing
__ProjectFeeTreasury_init_unchained	Internal	Can Modify State	onlyInitializing
_verifyMerkleData	Internal	-	-
_transferOut	Internal	Can Modify State	-

4.3 Vulnerability Summary

[N1] [Suggestion] Administrative Role Removal May Leave Roles Unassigned

Category: Authority Control Vulnerability Audit

Content

The AccessManager contract allows role members to remove themselves through `renounceRole`, while authorized role administrators may remove members through `revokeRole`. If the final holder of an administrative role is removed, that role may become unassigned until another authorized role can grant it again.

For example, removing the final holder of `DEFAULT_ADMIN_ROLE` may prevent routine role management. Similarly, if `UPGRADE_ROLE` is self-governed and its final holder is removed, future UUPS upgrades may require an alternative recovery mechanism. The practical impact depends on the configured role hierarchy and available recovery procedures. This is primarily an operational and configuration consideration rather than a direct permission bypass.

Code Location:

`src/Utils/AccessManager.sol#L93-106`

```
function revokeRole(bytes32 role, address account) external
onlyRole(_adminOf(role)) {
    ...
}

function renounceRole(bytes32 role) external {
    ...
}
```

Solution

Document the minimum-holder and recovery requirements for critical roles. Consider using multisig-controlled role holders, monitoring critical role changes, and testing the complete role hierarchy. If unassigned critical roles are unacceptable, add minimum-holder checks to renounceRole and revokeRole, or provide a separately governed recovery mechanism.

Status

Fixed

[N2] [Suggestion] Lack of Target Role Holder Validation When Setting Role Administrators

Category: Authority Control Vulnerability Audit

Content

The setRoleAdmin and batchSetRoleAdmin functions in the AccessManager contract are used to update the administrator role (adminRole) that governs a specific role. During the update process, the code does not verify whether the target adminRole currently has at least one valid holder. If the administrator is mistakenly set to an unoccupied role (empty role), permission management for the controlled role will fall into deadlock, and no party will be able to grant or revoke the controlled role thereafter.

Code Location:

src/Utils/AccessManager.sol

```
function setRoleAdmin(bytes32 role, bytes32 adminRole) external
onlyRole(_adminOf(role)) {
```

```

    ...
}

function batchSetRoleAdmin(bytes32[] calldata roles, bytes32 adminRole) external {
    ...
}

```

Solution

Add validation logic prior to updating the administrator role to ensure the target adminRole has at least one holder at present. Alternatively, adopt a secure pending-accept (two-step confirmation) pattern, whereby holders of the target adminRole proactively confirm and accept the administrative privileges.

Status

Fixed

[N3] [Suggestion] Redundant Conditional Logic in the _adminOf Function

Category: Others

Content

The internal _adminOf function in the AccessManager contract is used to query the administrator role that governs a specific role. The function includes a conditional check if (adminRole == bytes32(0)), which returns AccessRoles.DEFAULT_ADMIN_ROLE if the condition evaluates to true. Since the constant hash value of DEFAULT_ADMIN_ROLE is inherently bytes32(0), this conditional branch and value replacement are logically redundant. It introduces unnecessary contract bytecode and raises Gas costs upon execution.

Code Location:

src/Utils/AccessManager.sol#L137-142

```

function _adminOf(bytes32 role) internal view returns (bytes32 adminRole) {
    adminRole = roleAdmin[role];
    if (adminRole == bytes32(0)) {
        return AccessRoles.DEFAULT_ADMIN_ROLE;
    }
}

```

```
    }  
}
```

Solution

Remove this redundant if conditional block and directly return the value retrieved from the roleAdmin[role] mapping.

Status

Fixed

[N4] [Medium] Missing Core Attribute Validation When Setting NAV Oracle

Category: Design Logic Audit

Content

The setNAVOracle function in the Accountant.sol contract updates the address of the NAV oracle utilized by the accountant. When configuring a new navOracle, the function only verifies whether the address corresponds to a contract (oracle.code.length == 0), yet fails to validate whether the core attributes of the newly configured oracle align with the existing system.

Specifically, it does not check if the new oracle's vault() returns the contract's coreVault, nor whether the new oracle's baseAsset() matches the asset() within the coreVault. If an administrator mistakenly configures an incompatible oracle, incorrect pricing of underlying assets will occur. This distorts the conversion ratios for convertToShares and convertToAssets, potentially triggering severe accounting inconsistencies or financial losses from arbitrage exploits.

Code Location:

src/core/Accountant.sol#L65-70

```
function setNAVOracle(address oracle) external  
onlyRole(AccessRoles.VAULT_OPERATOR_ROLE) {  
    if (oracle.code.length == 0) revert InvalidNAVOracle(oracle);  
    address oldOracle = navOracle;  
    navOracle = oracle;  
    emit NAVOracleUpdated(oldOracle, oracle);  
}
```

Solution

Implement parameter validation logic prior to executing `navOracle = oracle;`. Invoke the relevant interfaces to verify that the new oracle's vault attribute matches the current `coreVault`, and confirm its `baseAsset` attribute is identical to the asset of the `coreVault`.

Status

Acknowledged; The project team responded that Oracle is a pluggable component and is maintained by a multisignature wallet.

[N5] [Medium] Lack of Symmetry Coverage Check for Adapter Position Sources

Category: Design Logic Audit

Content

In the `_validateRegistryAssetCoverage` function of the `AssetSummedNAVOracle.sol` contract, this internal function ensures every non-base asset supported by the registry has a corresponding valuation path prior to NAV computation.

`registerPositionAsset` (asset registration) and `registerPositionSource` (position source registration) are two separate configuration functions. The current logic lacks a symmetric check to verify whether value-bearing adapters exist within the `_positionSources` list.

If an adapter is enabled and configured to `reportsPositions`, yet omitted from `_positionSources` due to oversight or misconfiguration, the oracle will exclude assets held within this adapter when calculating `totalAssets`. This will result in substantial understatement of the system's total assets and NAV.

Code Location:

`src/core/NAVOracle/AssetSummedNAVOracle.sol#L306-316`

```
function _validateRegistryAssetCoverage(address baseAsset_) internal view {
    address[] memory assets = IAdapterRegistry(adapterRegistry).supportedAssets();
    for (uint256 i; i < assets.length; ++i) {
        address token = assets[i];
        if (token == baseAsset_) continue;
    }
}
```

```

        if (positionAssetConfig[token].adapterId == bytes32(0)) {
            revert UnsupportedRegistryPositionAsset(token);
        }
    }
}

```

Solution

Add symmetric validation logic inside the `_validateRegistryAssetCoverage` function: iterate over the adapter registry and guarantee that every registered adapter marked as enabled and reportsPositions is properly registered and included in the oracle's `_positionSources` list.

Status

Fixed

[N6] [Suggestion] Slippage Check Fails to Account for Deflationary Token Impacts

Category: Design Logic Audit

Content

The `_claimRedeem` function within the `JtrsyAdapter`, `ERC7540Adapter`, `HybridSyncDepositAsyncRedeemAdapter` and `VRPCQAdapter` contracts handles the withdrawal of redemption assets and enforces the minimum received amount (slippage) verification. The current code conducts slippage checks directly using the assets value returned by the redemption method. If the underlying asset is a deflationary token (e.g., tokens with transfer taxes or burn mechanisms), the actual token balance transferred to the recipient address will be lower than the returned assets quantity.

This flaw allows the slippage check to pass based on an inflated, inaccurate value. As a result, the protocol or users receive fewer assets than anticipated without the transaction being blocked in a timely manner, creating tangible risks of financial loss.

Code Location:

`src/adapters/ERC7540Adapter.sol#L146`

src/adapters/HybridSyncDepositAsyncRedeemAdapter.sol#L107

src/adapters/JtrsyAdapter.sol#L152

src/adapters/VRPCQAdapter.sol#L167

```
function _claimRedeem(TradePlan calldata plan) internal returns (AdapterResult
memory result) {
    ...

    _requireMinAmountOut(plan.minAmountOut, assets);

    ...
}
```

Solution

Record the asset balance of the recipient address immediately before and after the token transfer operation. Pass the difference between these two balance readings (the actual amount received) as the input parameter to the _requireMinAmountOut function for slippage validation.

Status

Acknowledged; The project team responded that they have verified the output of all adapters at the unified execution layer based on the actual balance received; the protocol does not support deflationary or transfer tax tokens.

[N7] [High] Incorrect Withdrawal Queue Query Address Causes Omission in Unsettled Asset

Calculation

Category: Design Logic Audit

Content

The _pendingWithdrawalAssets function in the VRPCQAdapter contract aggregates assets tied to incomplete requests in the external withdrawal queue, ensuring these outstanding external unsettled assets are incorporated into the Net Asset Value (NAV) calculation. When fetching the list of withdrawal requests, the code incorrectly passes the escrow address as the query parameter: getWithdrawalRequests(escrow).

Per the contract logic and labeling, the adapter contract itself is the entity that initiates withdrawal requests, meaning the queue requests belonging to the current contract should be queried instead. This bug prevents pending withdrawal assets of the current contract from being properly retrieved and summed. Consequently, the overall NAV is undercalculated, undermining the accuracy of associated share pricing and financial metrics.

Code Location:

src/adapters/VRPCQAdapter.sol#L190

```
function _pendingWithdrawalAssets(address vault) internal view returns (uint256
total) {
    ...

    uint256[] memory requestIds =
    IWithdrawQueue(queue).getWithdrawalRequests(escrow);
    if (requestIds.length == 0) return 0;

    ...
}
```

Solution

Change the query target of the withdrawal queue from the escrow address to the current contract itself. Revise the query code to:

```
IWithdrawQueue(queue).getWithdrawalRequests(address(this))
```

Status

Fixed

[N8] [Suggestion] Missing Validation of Actual Deposited Shares for Asynchronous Redemption

Requests

Category: Design Logic Audit

Content

The _requestRedeem function in the CoreVaultAsync contract locks users' shares in the Escrow contract and creates

asynchronous redemption requests. When transferring shares via the call `Escrow(escrow).depositShares`, the code disregards the actual deposited amount returned by this function and does not verify whether the shares successfully deposited into Escrow match the input shares parameter.

The `depositShares` function calculates and returns the actual received shares based on the balance difference internally. If the actual received amount ends up lower than the input shares for any reason (such as deflationary token mechanisms), the request manager will still record the redemption request using the inflated original shares value. This results in the system recording an artificially inflated volume of pending redemption shares, creating a discrepancy between the Escrow ledger and real underlying assets. In subsequent request fulfillment processes, insufficient share balances may trigger transaction reverts or bad debt.

Code Location:

`src/core/vault/CoreVaultAsync.sol#L349`

```
function _requestRedeem(uint256 shares, address controller, address owner_)
internal returns (uint256 requestId) {
    ...

    Escrow(escrow).depositShares(owner_, shares, EscrowBuckets.LOCKED_SHARES);

    ...
}
```

Solution

Capture the return value from `Escrow(escrow).depositShares` and implement strict validation logic to confirm the actual deposited shares equal the expected shares; revert the transaction immediately if a mismatch is detected.

Alternatively, reorder the workflow to complete the share transfer first, then record the redemption request using the actual returned share quantity.

Status

Fixed

[N9] [Medium] Lack of Consistency Check Between Target Vault Underlying Assets and Share Tokens**Category: Design Logic Audit****Content**

The `_deployERC4626` and `_deployVaultAdapter` functions within the `AdapterManagerInstaller` contract deploy and register vault adapters complying with various standards (including ERC4626, ERC7540, etc.). During deployment and configuration, the code directly references the product asset address (`productAsset`) and the input `config.token` without verifying whether the actual underlying asset of the target vault (`targetVault`) matches `productAsset`. Additionally, `_deployVaultAdapter` omits validation to confirm that `config.token` is indeed the share token corresponding to the target vault.

Misconfiguration of parameters by administrators will create a mismatch between the deployed adapter and the target vault's underlying assets or share tokens. This can trigger transaction reverts during subsequent fund transfers, deposits, or redemptions, and may even impact subsequent price calculations and net asset value (NAV) computations by the price oracle.

Code Location:

`src/factory/installers/AdapterManagerInstaller.sol#L162-246`

```
function _deployERC4626(bytes32 productId, SupportedAssetInstallConfig memory
config)
    internal
    returns (IProductFactory.Component memory component)
{
    bytes32 adapterId = _adapterId(productId, config.adapterId, "erc4626");
    address[] memory supportedAssets = _adapterAssets(_productAsset(productId),
config.targetVault);

    ...
}

function _deployVaultAdapter(
    bytes32 productId,
    SupportedAssetInstallConfig memory config,
    string memory suffix,
```

```

        bytes32 implementationComponentId,
        bytes32 kind
    ) internal returns (IProductFactory.Component memory component) {
        bytes32 adapterId = _adapterId(productId, config.adapterId, suffix);
        address asset = _productAsset(productId);
        address[] memory supportedAssets = _adapterAssets(asset, config.token);

        ...

        _adapterMetadata(
            kind, config.targetVault, config.token, address(0), supportedAssets,
            _kindReportsPositions(kind)
        )
    );

    ...
}

```

Solution

Add strict pre-deployment consistency validation logic before adapter deployment.

For instance, insert the check `require(IERC4626(targetVault).asset() == productAsset)` inside `_deployERC4626`. For `_deployVaultAdapter`, implement dual validation to verify that both the vault's `asset()` address and its share token address align with the input parameters.

Status

Fixed

[N10] [Low] Risk of Collision Conflicts in Default Adapter ID Generation Logic

Category: Others

Content

The `_adapterId` function in the `AdapterModuleInstaller` contract generates default adapter identifiers for adapters without a custom configuration ID specified. The current default generation logic solely relies on the `productId` and a suffix denoting the adapter type (e.g., "erc4626", "erc7540").

When multiple underlying vaults of the same type (identical suffix) need to be integrated under one product (same

productId), this logic will produce identical adapterId values. This triggers identifier collisions during adapter registration calls to the AdapterRegistry. Subsequent adapter configurations will overwrite prior ones or cause registration transactions to fail, breaking the product's multi-asset scalability.

Code Location:

src/factory/installers/AdapterModuleInstaller.sol#L311-314

```
function _adapterId(bytes32 productId, bytes32 configuredId, string memory suffix)
internal pure returns (bytes32) {
    if (configuredId != bytes32(0)) return configuredId;
    return keccak256(abi.encodePacked(productId, ".adapter.", suffix));
}
```

Solution

Introduce a unique anti-collision factor (such as the iteration index or the underlying token address) into the default adapterId generation algorithm. Rewrite the generation logic to resemble keccak256(abi.encodePacked(productId, ".adapter.", suffix, index)), guaranteeing globally unique adapter IDs for every adapter under the same product.

Status

Fixed

[N11] [Suggestion] Redundant Iteration in Enabled Adapter Counting Logic

Category: Others

Content

The _enabledAdapterCount function of the AdapterModuleInstaller contract calculates the length of the adapter component array to be allocated prior to installation. When counting Topnod-type adapters, the code additionally calls _anyEnabled(config, AdapterType.Topnod) after the main loop completes. This internal call triggers a second full iteration over the entire asset array to verify the enabled status. Since all assets have already been traversed in the preceding for loop, this separate secondary check introduces unnecessary computational overhead and redundant logic.

Code Location:

src/factory/installers/AdapterManagerInstaller.sol#L307

```
function _enabledAdapterCount(AdapterModuleConfig memory config) internal pure
returns (uint256 count) {
    for (uint256 i = 0; i < config.supportedAssets.length; i++) {
        if (config.supportedAssets[i].enabled &&
!_isSharedAdapterType(config.supportedAssets[i].assetType)) {
            count++;
        }
    }
    if (_anyEnabled(config, AdapterType.Topnod)) count++;
}
```

Solution

Optimize the counting logic. During a single pass of the config.supportedAssets for loop, simply check whether config.supportedAssets[i].enabled evaluates to true, and increment the count variable accordingly if the condition holds. This eliminates the call to the _anyEnabled function and removes the secondary iteration entirely.

Status

Fixed

[N12] [Suggestion] Self-Managed Upgrade Permissions Introduce Critical Security Risks

Category: Authority Control Vulnerability Audit

Content

The _handoffProductAccess function in the ProductFactory contract transfers access control roles upon product finalization. The code executes the statement access.setRoleAdmin(AccessRoles.UPGRADE_ROLE, AccessRoles.UPGRADE_ROLE), which sets the administrative role for UPGRADE_ROLE to itself.

This means any address holding the UPGRADE_ROLE permission can not only perform contract upgrades but also independently grant or revoke upgrade privileges to other accounts. This self-administered permission pattern violates the Principle of Least Privilege and separation of duties. If the private key associated with the upgrade

permission is compromised, attackers can tamper with contract logic, gain full control over the upgrade permission group, and even remove legitimate original upgraders. This may result in the protocol losing overall control of its entire system.

Code Location:

src/factory/ProductFactory.sol#L601

```
function _handoffProductAccess(  
    address admin,  
    address vaultOperator,  
    address vaultCurator,  
    address vaultKeeper,  
    address upgradeAuthority,  
    address productAccessManager  
) internal {  
    ...  
  
    access.setRoleAdmin(AccessRoles.UPGRADE_ROLE, AccessRoles.UPGRADE_ROLE);  
  
    ...  
}
```

Solution

Self-management of the UPGRADE_ROLE is strongly discouraged. The administrative role of UPGRADE_ROLE should remain assigned to DEFAULT_ADMIN_ROLE, or be set to an independent role controlled by a Timelock contract or Governance Multisig. This improves security and transparency around all critical permission transfer workflows.

Status

Fixed

[N13] [Suggestion] Risk of Excessive Total Fee Rate Due to Stacked Multiple Fee Modules

Category: Design Logic Audit

Content

The `_runBeforeActionModules` function within the `FeeRouter` contract sequentially executes all fee modules attached to the same hook point (e.g., `BEFORE_DEPOSIT` or `BEFORE_REDEEM`) and aggregates the fee settlement outputs returned by each module via `cumulative = _addDelta(cumulative, moduleDelta)`.

The `FeeRouter` contract permits up to 16 fee modules to be attached to a single hook point, yet the system lacks an upper limit check on the aggregated total fees. If multiple modules (such as subscription fee modules, management fee modules, etc.) are configured with high individual rates, their combined total fee rate could exceed 100%. In such cases, the deducted fees would surpass the user's original principal or shares deposited or redeemed. This will trigger arithmetic overflows or transfer failures due to insufficient assets or shares during subsequent execution, inadvertently blocking core business workflows including deposits and withdrawals.

Code Location:

`src/core/FeeRouter.sol#L243`

```
function _runBeforeActionModules(bytes32 point, address caller, bytes calldata
input)
    internal
    returns (FeeDelta memory cumulative)
{
    address[] memory modules = _modulesByPoint[point];
    uint256 len = modules.length;

    for (uint256 i = 0; i < len; ++i) {
        FeeResult memory moduleResult = IFeeModule(modules[i]).beforeAction(point,
caller, input);
        FeeDelta memory moduleDelta = _settleModuleResult(point, input,
modules[i], moduleResult);
        IFeeModule(modules[i]).onFeeSettled(point, caller, input, moduleDelta);
        cumulative = _addDelta(cumulative, moduleDelta);
    }

    emit FeeModulesExecuted(vault, point, len);
}
```

Solution

Add a total fee cap validation during fee accumulation. Either inside the loop of `_runBeforeActionModules` or after the loop completes, verify that the aggregated `cumulative.assetOut` does not exceed the requested input asset amount, and that `cumulative.sharesTransferred` does not exceed the requested input share quantity. If the threshold is breached, either revert the transaction with descriptive error messages or cap the total fees at a safe value no greater than the original input amount.

Status

Fixed; Updated: Perform validation in an external vault.

[N14] [Suggestion] Forced Third-Party Reward Claim May Cause Permanent Fund Lockup

Category: Design Logic Audit

Content

The claim function in the `ProjectFeeTreasury` contract verifies Merkle proofs and disburses accrued rewards to the designated user address (`user`). Since there is no access validation on the transaction initiator (`msg.sender`), any external attacker or bot may submit a claim transaction on behalf of any target user once they obtain the user's corresponding Merkle proof.

While funds will ultimately be transferred solely to the predefined user address and cannot be directly stolen by attackers, this forced claiming mechanism introduces two critical risks:

It eliminates user flexibility to securely withdraw funds to custom recipient addresses via the combination of `verify` and `claimVerified(receiver)`.

If the target user is a smart contract lacking logic to withdraw the relevant token (such as multisig wallets or proxy contracts incompatible with the token standard), forced token transfers will permanently trap the tokens within the contract, resulting in irreversible asset losses.

Code Location:

`src/utis/ProjectFeeTreasury.sol#L42-55`

```
function claim(address user, uint256 totalAccrued, bytes32[] calldata proof)
    external
    override
    nonReentrant
    returns (uint256 amountOut)
{
    ...
}
```

Solution

Add identity validation within the claim function to restrict direct withdrawal operations exclusively to transactions where `msg.sender == user`. An alternative approach is to implement granular configuration controls that allow users to pre-set a default recipient address for automatic reward withdrawals.

Status

Fixed

[N15] [Suggestion] Single Price Oracle Source Lacks Cross-Verification Mechanism

Category: Others

Content

The `updatePrice` function within the `OffchainAssetPriceOracle` contract receives asset prices submitted by offchain Keeper nodes and updates the global system price feed. The current logic solely relies on a single source—accounts holding the `PROJECT_KEEPER_ROLE` permission—to submit price data, with no cross-verification across multiple oracle feeds.

If the private key of this single node is compromised, the node suffers downtime, or it is maliciously manipulated, attackers can submit arbitrary extreme prices drastically deviated from real market values to the contract. Since the protocol directly consumes this unvalidated price for calculations such as Net Asset Value (NAV), this flaw may trigger incorrect share pricing, malicious arbitrage opportunities, and massive financial losses.

Code Location:

src/utls/OffchainAssetPriceOracle.sol#L70-84

```
function updatePrice(address asset, uint256 price, bytes32 roundId)
    public
    onlyRole(AccessRoles.PROJECT_KEEPER_ROLE)
{
    ...

    _prices[asset] = PriceData({price: price, timestamp: block.timestamp,
roundId: roundId});

    emit PriceUpdated(asset, price, block.timestamp, roundId);
}
```

Solution

Implement a cross-verification mechanism with multiple independent nodes or third-party oracle providers (e.g., Chainlink, Pyth). Allow several isolated price feed nodes to submit price data, then apply aggregation algorithms such as Median or Time-Weighted Average Price (TWAP) within the contract to filter out outliers and compute the final reference price. Meanwhile, introduce an upper bound check for single-update price deviation. Any price update failing timeliness or sanity checks shall be rejected and reverted immediately to block abnormal price inputs.

Status

Acknowledged

[N16] [Suggestion] Asset Lockup Risk Arising from Conflicts Between Product Lockup Period and Redemption Window

Category: Design Logic Audit

Content

The `_setProductLockConfig` function in the `ProductLockRule` contract is used to configure the product lockup period, the time window during which redemptions are prohibited. When updating lockup configurations, the code only verifies that the lockup end timestamp is later than the lockup start timestamp. It lacks validation to check whether

the lockup period overlaps with the product's redemption window.

Improper administrator configuration may cause the lockup period to fully or partially cover the redemption window, or the entire redemption window to fall within the lockup timeframe. In such cases, users cannot submit redemption transactions during the designated eligible window, resulting in unintended asset lockup and severely impairing capital liquidity.

Code Location:

src/policy/lifecycle/ProductLockRule.sol#L86-92

```
function _setProductLockConfig(ProductLockConfig memory config) internal {  
    if (config.productLockEndTime < config.productLockStartTime) {  
        revert InvalidProductLockConfig(config.productLockStartTime,  
config.productLockEndTime);  
  
        ...  
    }  
}
```

Solution

Add cross-validation logic against redemption window parameters when configuring product lockup periods. Enforce that the entire lockup timeframe falls outside the redemption window by strictly validating either that the lockup end time precedes the redemption start time, or the lockup start time comes after the redemption end time.

Status

Fixed; Updated: The redemption period is separate and will take precedence when the lock-up period and redemption period overlap.

[N17] [Suggestion] Unbounded Policy Module Array Triggers Gas DoS on Hook Execution

Category: Denial of Service Vulnerability

Content

Within the VaultPolicyRouter contract's mountPolicy function, accounts holding the VAULT_OPERATOR_ROLE permission may continuously append policy modules to modulesByPoint[point], yet no maximum length constraint is

imposed on this array. Subsequent `beforeAction` and `afterAction` logic traverses and invokes all active modules stored in the array via `_dispatchBefore` and `_dispatchAfter`. When the array accumulates an excessive number of modules, core workflows including deposits, mints, redemptions, withdrawals, liquidations and other transactions risk reverting due to exceeding the block gas limit. The `unmountPolicy` function also requires a linear scan of the array; oversized arrays drastically raise the gas overhead required to remove modules.

Code Location:

`src/core/VaultPolicyRouter.sol#L57-71`

```
function mountPolicy(address subject, bytes32 point, address module)
    external
    onlyRole(AccessRoles.VAULT_OPERATOR_ROLE)
{
    ...

    store.modulesByPoint[point].push(module);

    ...
}
```

Solution

Enforce a hard maximum limit for policy modules per subject and hook point (e.g., a constant named `MAX_POLICY_MODULES_PER_POINT`). Validate the array length prior to appending new modules in `mountPolicy`, and revert the transaction if the cap would be exceeded.

Status

Fixed

[N18] [Low] Subscription Whitelist Validates Payer Instead of Beneficiary

Category: Design Logic Audit

Content

Within the `SubscriptionWhitelistGuard` contract, the `decodeRequestDeposit` function returns values in the sequence

(assets, controller, owner). However, the guard only extracts the third output parameter owner for whitelist validation.

owner refers to the payer, while controller is the entity whose subscription rights are recorded in RequestManager and subject to authorization checks during claim operations.

A whitelisted user may call requestDeposit and assign a non-whitelisted address as the controller, enabling that unlisted address to later claim the corresponding shares.

Additionally, CoreVaultAsync._claim does not invoke any policy hooks, leaving the entire claim workflow without whitelist validation. If this whitelist mechanism enforces KYC, AML, or jurisdictional restrictions, access controls can be fully bypassed.

Code Location:

src/policy/guards/SubscriptionWhitelistGuard.sol#L57-71

```
function beforeAction(  
    bytes32 point,  
    address caller,  
    bytes calldata input  
) external view override {  
    _requirePolicyRouter();  
    address account = caller;  
    if (point == BEFORE_REQUEST_DEPOSIT) {  
        (, , account) = VaultHookCodec.decodeRequestDeposit(input);  
    }  
    _requireWhitelisted(account);  
}
```

Solution

Clarify the subject of whitelist restrictions per business logic and validate at minimum the controller. If both payers and beneficiaries must comply with regulations, validate owner, controller, and receiver simultaneously. Add policy hook validation for claim workflows such as deposit and mint operations.

Status

Fixed; Updated: The method for claiming asynchronous deposits and redemptions has been changed so that the corresponding assets are directly transferred to the owner during settlement by the keeper.

[N19] [Medium] Retrospective Fee Charging Caused by Improper Initialization of Performance Fee

High-Water Mark

Category: Design Logic Audit

Content

In the HWMPPerformanceFee contract, when performance fees are enabled for the first time or the high-water mark equals zero, the contract directly assigns the configured hurdleNavPerShare to highWaterMark without fetching the current NAV. If the vault has already generated historical gains exceeding the hurdle rate, or if the fee rate is switched from zero to a non-zero value, the subsequent CRYSTALLIZE operation will include all pre-enablement gains in the performance fee calculation. Given that fees are currently settled via minting new shares, overcharged fees will permanently dilute existing share holders.

Code Location:

src/fee/HWMPPerformanceFee.sol#L244

```
function _setPerformanceConfig(
    ...

    if (
        config.performanceFeeBps > 0 &&
        (!wasConfigured || _state.highWaterMark == 0)
    ) {
        _state.highWaterMark = config.hurdleNavPerShare;
    }

    ...
}
```

Solution

When enabling performance fees for the first time or resetting the high-water mark, set the watermark value to

max(_currentNavPerShare(), hurdleNavPerShare).

Status

Fixed

[N20] [Medium] Vault Operations Freeze When Management Fee Liabilities Exceed Current Assets

Category: Design Logic Audit

Content

Within the TimeAccrualFee contract, accrued management fee liabilities are converted into shares via

NAVFeeMath.feeAssetsToMintShares. This function reverts immediately if feeAssets >= accounting.totalAssets or the total share supply equals zero.

uncollectedAssets is calculated based on the historical lastTotalAssets, while accounting.totalAssets aggregates assets held in the escrow contract plus the value of positions the adapter holds in external vaults. This edge case can be triggered if an external vault incurs negative net asset value due to market volatility or exploits.

The management fee module is attached to multiple BEFORE_* hooks. Any price feed revert will break end-to-end workflows including deposits, redemptions, asynchronous fulfillments and fee crystallization, while historical state values cannot be updated, resulting in a vault-wide Denial-of-Service (DoS).

Code Location:

src/fee/TimeAccrualFee.sol

```
function _settleElapsed() internal returns (uint256 feeAssets) {
    ...

    uint256 totalAssetsBasis = state.lastTotalAssets;
    uint16 bps = _config.annualFeeBps;
    if (timeElapsed != 0 && totalAssetsBasis != 0 && bps != 0) {
        uint256 annualFeeAssets = Math.mulDiv(
            totalAssetsBasis,
            uint256(bps),
            10_000
        );
        feeAssets = Math.mulDiv(
```

```

        annualFeeAssets,
        timeElapsed,
        SECONDS_PER_YEAR
    );
    state.uncollectedAssets += feeAssets;
}
}

function _previewCrystallize(
    IAccountant.AccountingSnapshot memory accounting,
    uint256 feeAssets
) internal view returns (FeeResult memory) {
    ...

    uint256 feeShares = cfg.settlementType == FeeSettlementType.MINT
        ? NAVFeeMath.feeAssetsToMintShares(
            vault,
            feeAssets,
            accounting.totalAssets,
            accounting.totalSupply
        )
        : 0;
    uint256 amount = cfg.settlementType == FeeSettlementType.TRANSFER
        ? feeAssets
        : feeShares;

    ...
}

```

src/libraries/NAVFeeMath.sol#L68-89

```

function feeAssetsToMintShares(
    address vault,
    uint256 feeAssets,
    uint256 totalAssets,
    uint256 totalSupply,
    Math.Rounding rounding
) internal pure returns (uint256 shares) {
    if (feeAssets == 0) return 0;
    if (totalSupply == 0) revert NAVFeeSupplyUnavailable(vault, totalSupply);
    if (feeAssets >= totalAssets) revert NAVFeeAssetsExceedTotalAssets(vault,
feeAssets, totalAssets);
}

```

```
        return Math.mulDiv(feeAssets, totalSupply, totalAssets - feeAssets,
rounding);
    }
```

Solution

Implement an additional privileged function that exclusively invokes `_syncBasis` to refresh historical state under such unexpected scenarios, eliminating the risk of permanent DoS.

Status

Fixed

[N21] [Low] The Adapter Enable Toggle Does Not Cover Critical Paths

Category: Design Logic Audit

Content

Within the `AdapterRegistry` contract, `setAdapterEnabled` only updates `adapters[adapterId].enabled`. This status flag is exclusively validated by the `requireAdapterFor` and `requireAdapter` functions. However, the `AssetSummedNAVOracle` contract caches all adapter addresses upon registration. Subsequent valuation calculations directly invoke the `positions()` method on these cached addresses without rechecking the enabled status.

Likewise, the `ACModule` contract uses `adapterOf` to authenticate callers for asynchronous settlement operations, and `adapterOf` does not verify whether the target adapter has been disabled. As a result, after an adapter is disabled, its positions may still be included in NAV calculations, and asynchronous settlement operations against it remain executable.

Code Location:

`src/core/AdapterRegistry.sol`

```
function getAdapterInfo(bytes32 adapterId) external view returns (AdapterInfo
memory) {
    ...
}

function adapterOf(bytes32 adapterId) external view returns (address adapter) {
```

```

    ...
}

```

Solution

Implement a unified read interface `requireEnabledAdapter`, and apply it in both the NAV valuation workflow and asynchronous settlement authorization paths.

During every NAV valuation, re-validate the registration status of adapters. If an adapter is detected as disabled, return an invalid state or revert immediately, and refrain from utilizing cached adapter records.

Status

Acknowledged; The project team responded: The ban is only to close new risk exposures; existing position valuations and asynchronous claims must continue.

[N22] [Suggestion] Improper Upper Bound Check for Withdrawal Exit Fee

Category: Others

Content

Within the `BEFORE_WITHDRAW` branch of the `ExitFixedBpsFee` contract, the exit fee is calculated as: $\text{netShares} * \text{feeBps} / (10_000 - \text{feeBps})$. When `feeBps` approaches 10,000, the multiplier grows extremely large. The current implementation permits a maximum value of 9,999 bps.

The `CoreVaultSync` contract directly adds `shareFee` to the shares owed by the user, without the same `ExcessiveExitFee` upper limit check present in the `_redeem` logic.

Consequently, a fee of 5,000 bps forces the user to pay approximately twice the base shares, while 9,999 bps can demand roughly 10,000 times the base shares. The withdrawal transaction will fail if the user holds insufficient balance.

Code Location:

`src/fee/ExitFixedBpsFee.sol#L48-53`

```

function _feeAmount(bytes32 point, bytes calldata input, uint16 feeBps, FeeDelta
memory prior)

```

```

        internal
        view
        override
        returns (uint256)
    {
        if (point == BEFORE_REDEEM) {
            ...
        } else if (point == BEFORE_FULFILL_REDEEM) {
            ...
        } else if (point == BEFORE_WITHDRAW) {
            (uint256 assets,,) = VaultHookCodec.decodeWithdraw(input);
            uint256 netShares = _withdrawNetShares(assets, prior);

            return Math.mulDiv(netShares, feeBps, 10_000 - feeBps,
Math.Rounding.Ceil);
        }
        revert InvalidFixedBpsFeeContext();
    }

```

Solution

It is recommended to tighten ExitFixedBpsFee._validateFeeBps to a practical fee range permitted by the protocol, for example, capping it at 1,000 bps.

Status

Acknowledged; The project team responded that the exit fee is a product configuration parameter controlled by a specific role, and the actual fee rate will be controlled within a reasonable range through manual review and multi-signature approval when the product is launched and parameters are changed.

[N23] [Suggestion] No Upper Bound on Supported Asset List

Category: Others

Content

In the AdapterRegistry contract, _supportedAssets simply appends new assets upon first enabling support for them without any cardinality limit. The AssetSummedNAVOracle contract enforces a hard limit of 50 positionAsset entries. Each invocation of totalAssets() within AssetSummedNAVOracle requires every non-base asset marked as supported

in AdapterRegistry to have a corresponding PositionAssetConfig[token].

Once the number of non-base assets in AdapterRegistry exceeds this capacity limit, configurations cannot be created for excess assets, resulting in persistent reverts during NAV validation. Even before hitting the capacity ceiling, the window between adding asset support and synchronizing the corresponding position configuration triggers reverts in totalAssets(). This disrupts deposits, redemptions, settlements and fee processing. Furthermore, iterating over an oversized asset list in full may introduce Gas Denial-of-Service (Gas DoS) risks.

Code Location:

src/core/AdapterRegistry.sol#L168-184

```
function _setSupportedAsset(bytes32 adapterId, address asset, bool supported)
internal {
    ...
}
```

Solution

Enforce an upper bound on _supportedAssets no greater than the downstream MAX_POSITION_ASSETS, and restrict the input array length of registerAdapter.

Alternatively, atomically update asset support status alongside NAV position configurations: only permit an asset to be added to the global supported list after the corresponding positionAssetConfig has been successfully created.

Status

Fixed

[N24] [Medium] Excessive Privilege Risks

Category: Authority Control Vulnerability Audit

Content

1. In the registerInstaller function of the ProductFactory contract, accounts holding PROJECT_OPERATOR_ROLE may register any Installer that passes the moduleId() check. Subsequent installModules operations invoke the registered Installer via delegatecall through ProductFactory, executing code within the storage context of the factory

contract. Accordingly, a malicious or unaudited Installer may alter critical state variables including factory permissions, implementation addresses, and product component records. This can disrupt future product deployments and even compromise the entire permission model of the factory.

Code Location:

src/factory/ProductFactory.sol#L124-138

```
function registerInstaller(bytes32 moduleId, address installer)
    external
    override
    onlyRole(AccessRoles.PROJECT_OPERATOR_ROLE)
{
    ...
}
```

2. In `registerImplementation` and `registerImplementations` functions of the `ProductFactory` contract, holders of `PROJECT_OPERATOR_ROLE` are authorized to overwrite the latest implementation address for any component. Installers retrieve this latest implementation to deploy UUPS or other proxy-based components. A malicious implementation may execute unauthorized logic during initialization or subsequent calls, impacting newly created products and core modules installed thereafter. This permission does not alter already deployed proxies immediately, yet it poisons the deployment pipeline for all future instances.

Code Location:

src/factory/ProductFactory.sol#L141-163

```
function registerImplementation(
    bytes32 componentId,
    address target,
    IProductFactory.DeploymentModel model,
    bytes32 expectedCodeHash
) external override onlyRole(AccessRoles.PROJECT_OPERATOR_ROLE) {
    _registerImplementation(componentId, target, model, expectedCodeHash);
}
```

```
function registerImplementations(IProductFactory.ImplementationRegistration[]
calldata registrations)
    external
    override
    onlyRole(AccessRoles.PROJECT_OPERATOR_ROLE)
{
    ...
}
```

3. In the FeeRouter contract, accounts with VAULT_OPERATOR_ROLE are permitted to add fee modules to FeeRouter. The same role may insert, remove modules and configure typed fee modules. `_addFeeModule` imposes no requirement for module implementations to originate from a trusted whitelist, nor does it enforce independent governance review for fee recipient addresses and fee amounts returned by modules. Settlement outputs from modules trigger asset/share transfers and fee share minting via escrow within FeeRouter. A malicious module could cause user funds to be diverted as fees or mint excessive share tokens, diluting holdings of other users.

Code Location:

src/core/FeeRouter.sol#L75-85

```
function addFeeModule(bytes32 point, address module) external
onlyRole(AccessRoles.VAULT_OPERATOR_ROLE) {
    _addFeeModule(point, module, _modulesByPoint[point].length);
}

function addFeeModuleAt(bytes32 point, address module, uint256 index)
    external
    onlyRole(AccessRoles.VAULT_OPERATOR_ROLE)
{
    _addFeeModule(point, module, index);
}
```

4. In the Accountant contract, holders of VAULT_OPERATOR_ROLE may replace the NAV Oracle. The Accountant directly invokes `totalAssets()` on the newly assigned Oracle. A malicious Oracle can report artificially inflated or deflated asset values, which distorts NAV calculations and breaks share exchange logic for deposits, withdrawals and redemptions.

Code Location:

src/core/Accountant.sol#L65-70

```
function setNAVOracle(address oracle) external
onlyRole(AccessRoles.VAULT_OPERATOR_ROLE) {
    ...
}
```

5. In the OffchainAssetPriceOracle contract, PROJECT_KEEPER_ROLE holders may submit price data for any asset. If the keeper private key is compromised or misused, an adversary can manipulate prices for all assets consuming this Oracle. The AssetSummedNAVOracle relies on these prices for asset valuation, introducing risks of NAV manipulation, incorrect share minting and improper redemptions.

Code Location:

src/utils/OffchainAssetPriceOracle.sol#L70-84

```
function updatePrice(address asset, uint256 price, bytes32 roundId)
public
onlyRole(AccessRoles.PROJECT_KEEPER_ROLE)
{
    ...
}
```

6. In the ProjectFeeTreasury contract, VAULT_OPERATOR_ROLE holders can directly set new Merkle roots and distribution batch IDs. Should this role's private key leak or receive misconfiguration, an attacker can craft a malicious Merkle root containing their own address and arbitrary fund amounts, then withdraw funds from the Treasury via the claim function. Setting an invalid Merkle root may also block legitimate users from claiming entitlements.

Code Location:

src/utils/ProjectFeeTreasury.sol#L88-101

```
function setMerkleRoot(bytes32 newMerkleRoot, bytes32 newDistributionId)
    external
    override
    onlyRole(AccessRoles.VAULT_OPERATOR_ROLE)
{
    ...
}
```

Solution

In the short term, transferring the ownership of core roles to time-locked contracts and managing them by multi-signatures is an effective solution to avoid single-point risks. However, in the long run, a more reasonable solution is to implement a permission separation strategy and set up multiple privileged roles to manage each privileged function separately. Permissions involving user funds and contract core parameter updates should be managed by the community, while permissions involving emergency contract suspensions can be managed by EOA addresses. This ensures the safety of user funds while responding quickly to threats.

Status

Acknowledged

[N25] [Information] External Dependencies Disclosure

Category: Others

Content

1. Within the `_executeTrade` function of the `ACModule` contract, the system executes deposit trades via `IAdapter(adapter).execute(plan)`. The associated Adapter deposits assets into an external Vault. Accordingly, asset security relies on the implementation, state and administrator privileges of the external Vault. If the external Vault suffers logical vulnerabilities, malicious upgrades, pause-and-freeze actions, incorrect pricing, or solvency issues, users may face asset losses, inability to redeem, or inaccurate valuation of Adapter positions.
2. In the `_pendingWithdrawalAssets` function of the `VRPCQAdapter` contract, the Adapter fetches `WithdrawalRequestStatus` from an external queue and aggregates the `amountToWithdraw` of all incomplete requests

as pending withdrawal assets. This value is subsequently consumed by `positions()` for NAV valuation.

Asset value is therefore directly dependent on the request list, completion status and amounts returned by the external Queue. Bugs, malicious implementation upgrades, anomalous return data, or outages within the Queue contract may lead to overvalued or undervalued NAV, disrupting share minting, redemption amounts and liquidity assessments.

Code Location:

`src/adapters/VRPCQAdapter.sol#L186-199`

```
function _pendingWithdrawalAssets(address vault) internal view returns (uint256
total) {
    ...
}
```

3. In the `TopnodRouter` contract, `PendingOrder` only records the adapter and order kind, without capturing the token and amount corresponding to each `traceId`. `onOrderSettled` and `onBatchOrderSettled` functions merely verify that the caller is `topnodProxy`, then transfers funds from the Router's balance to the Adapter using externally supplied token and amount parameters.

If the configured `TopnodProxy` is compromised, contains implementation flaws, or returns corrupted data, it may specify arbitrary tokens and amounts against valid `traceId` values. Since the Router settles transactions using a shared balance, an attacker could siphon excess assets belonging to other orders or other products. Insufficient balance may also cause permanent settlement failures leaving orders indefinitely pending.

Code Location:

`src/adapters/routers/TopnodRouter.sol#L128-158`

```
function onBatchOrderSettled(
    bytes32 traceId,
    bool success,
    address[] calldata tokens,
    uint256[] calldata amounts,
```

```
        bytes calldata data
    ) external override {
        ...
    }

    function onOrderSettled(bytes32 traceId, bool success, address token, uint256
amount, bytes calldata data)
        external
        override
    {
        ...
    }
```

Solution

N/A

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002608040002	SlowMist Security Team	2026.07.06 - 2026.08.04	Medium Risk

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 high risk, 6 medium risk, 3 low risk vulnerabilities, 14 suggestions and 1 information. All the findings were fixed or acknowledged. Since the project has not yet been deployed to the mainnet and the permission of the core role has not yet been transferred, the risk level reported is temporarily medium.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.



Official Website
www.slowmist.com



E-mail
team@slowmist.com



Twitter
[@SlowMist_Team](https://twitter.com/SlowMist_Team)



Github
<https://github.com/slowmist>