SuperEarn - Audit

Security Assessment

CertiK Assessed on Feb 11th, 2026

CertiK Assessed on Feb 11th, 2026

SuperEarn - Audit

The security assessment was prepared by CertiK.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Vault	EVM Compatible	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE
Solidity	Preliminary comments published on 01/30/2026 Final report published on 02/11/2026

Vulnerability Summary

52	34	0	18	0
Total Findings	Resolved	Partially Resolved	Acknowledged	Declined

2	Centralization	2 Acknowledged	Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.
0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
1	Major	1 Acknowledged	Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
12	Medium	9 Resolved, 3 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
20	Minor	13 Resolved, 7 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
17	Informational	12 Resolved, 5 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | SUPEREARN - AUDIT

Optimizations

CODEBASE | SUPEREARN - AUDIT

Repository

https://github.com/superearn-io/superearn-core/tree/70fa63512fd794b17a55d13bc840243740ee063c

https://github.com/superearn-io/superearn-core/tree/b528806a89fb4956b09e32f3dfdf93926985b292

https://github.com/superearn-io/superearn-core/tree/6c7cfc54a5dbabd86ce753dfb255c36157236f71

https://github.com/superearn-io/superearn-core/tree/67fe7749ba53fa660ca763b07510eb298485e37b

https://github.com/superearn-io/superearn-core/tree/c53e2b02c7adff0b6f97e8b77a06cd69952a4395

https://github.com/superearn-io/superearn-core/tree/b4a97c94de074023161dbd223b12faa27dd6b289

Commit

70fa63512fd794b17a55d13bc840243740ee063c

b528806a89fb4956b09e32f3dfdf93926985b292

6c7cfc54a5dbabd86ce753dfb255c36157236f71

67fe7749ba53fa660ca763b07510eb298485e37b

c53e2b02c7adff0b6f97e8b77a06cd69952a4395

b4a97c94de074023161dbd223b12faa27dd6b289

Audit Scope

The file in scope is listed in the appendix.

APPROACH & METHODS | SUPEREARN - AUDIT

This audit was conducted for SuperEarn to evaluate the security and correctness of the smart contracts associated with the SuperEarn - Audit project. The assessment included a comprehensive review of the in-scope smart contracts. The audit was performed using a combination of Static Analysis, Formal Verification, and Manual Review.

The review process emphasized the following areas:

- Architecture review and threat modeling to understand systemic risks and identify design-level flaws.
- Identification of vulnerabilities through both common and edge-case attack vectors.
- Manual verification of contract logic to ensure alignment with intended design and business requirements.
- Dynamic testing to validate runtime behavior and assess execution risks.
- Assessment of code quality and maintainability, including adherence to current best practices and industry standards.

The audit resulted in findings categorized across multiple severity levels, from informational to critical. To enhance the project's security and long-term robustness, we recommend addressing the identified issues and considering the following general improvements:

- Improve code readability and maintainability by adopting a clean architectural pattern and modular design.
- Strengthen testing coverage, including unit and integration tests for key functionalities and edge cases.
- Maintain meaningful inline comments and documentations.
- Implement clear and transparent documentation for privileged roles and sensitive protocol operations.
- Regularly review and simulate contract behavior against newly emerging attack vectors.

REVIEW NOTES | SUPEREARN - AUDIT

Overview

SuperEarn is a Solidity-based, upgradeable cross-chain yield aggregation protocol deployed on multiple chains including Kaia and Ethereum. The protocol consists of two main components: `core` and `v2`. The `core` component provides foundational infrastructure for yield generation and asset management, while the `v2` component enables cross-chain yield aggregation. These components work together to allow users on the Kaia chain to deposit USDT, which is then bridged and deployed into vaults on Ethereum built using Yearn Finance code, enabling cross-chain asset management. Critical functions such as configuration updates, asset deployment, redemption processing, and emergency operations are controlled by privileged roles through on-chain role-based access control.

External Dependencies

The contracts are serving as the underlying entities to interact with one or more third party protocols, such as `Chainlink CCIP`, `@metamorpho`, `@yearn-vaults`. The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of third parties can possibly create severe impacts, such as causing incorrect price calculations, denial of service or fund loss.

The protocol integrates with the following third-party protocols and contracts:

- `yearnVault`: the vault address
- `IAny2EVMMessageReceiver`: the Chainlink CCIP interface used by `CrosschainAdapter` for cross-chain messaging
- `CCIPReceiverUpgradeable`: the upgradeable Chainlink CCIP receiver implementation used by `CrosschainAdapter` and `RunespearReceiver`
- `AggregatorV3Interface`: the Chainlink price feed interface used by `AssetPriceConverter` for price data
- `IFeedProxy`: the Orakl price feed interface used by `OraklAssetPriceConverter` for price data on chain
- `UniversalSwapRouter`: the Uniswap swap router used for token swaps
- `IMetaMorpho`: the MetaMorpho interface used by `StrategyMorphoV1Vault` for yield generation
- `BaseStrategy`: the Yearn base strategy contract inherited by strategy implementations
- `CommonHealthCheck`: the Yearn health check contract inherited by `HealthCheck`

The following libraries/contracts are considered as the third-party dependencies:

- `@openzeppelin/contracts-upgradeable/`
- `@openzeppelin/contracts/`
- `@chainlink/contracts-ccip/`
- `@yearn-vaults/`
- `@metamorpho/`

We assume these contracts or addresses are valid and non-vulnerable actors and implementing proper logic to collaborate with the current project.

Privileged Functions

In the **SuperEarn** project, privileged roles are adopted to ensure the dynamic runtime updates of the project, which are specified in the **Centralization Risks** findings.

The advantage of those privileged roles in the codebase is that the client reserves the ability to adjust the protocol according to the runtime requirements to best serve the community. It is also worth noting the potential drawbacks of these functions, which should be clearly stated through the client's action/plan. Additionally, if the private keys of the privileged accounts are compromised, it could lead to devastating consequences for the project.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of a `Timelock` contract.

FINDINGS | SUPEREARN - AUDIT

52	0	2	1	12	20	17
Total Findings	Critical	Centralization	Major	Medium	Minor	Informational

This report has been prepared for SuperEarn to identify potential vulnerabilities and security issues within the reviewed codebase. During the course of the audit, a total of 52 issues were identified. Leveraging a combination of Static Analysis, Formal Verification & Manual Review the following findings were uncovered:

ID	Title	Category	Severity	Status
SUA-47	**Centralized Control Of Contract Upgrade**	**Centralization**	**Centralization**	**Acknowledged**
SUA-48	**Centralization Related Risks**	**Centralization**	**Centralization**	**Acknowledged**
SUA-11	Fixed Request-Time `requestedAssets` Can Permanently Deadlock The Redemption Queue After Losses/Shortfalls	Logical Issue	Major	Acknowledged
SUA-01	Potential Share Loss When Claiming	Volatile Code	Medium	Acknowledged
SUA-02	Unclaimable Redemptions From Zero Asset Amounts	Volatile Code	Medium	Resolved
SUA-21	Over-Permissive Access To `sendAssets`	Access Control	Medium	Resolved
SUA-23	Incorrect Supply Cap Calculation Causes Underutilization Of Investment Capacity	Logical Issue	Medium	Resolved
SUA-24	Bridge Reconciliation Runs On Stale Snapshots	Logical Issue	Medium	Resolved
SUA-25	Incomplete Round-Based Protection	Logical Issue	Medium	Acknowledged
SUA-26	Incomplete Liquidation Due To Ignored Underlying Token Balance	Logical Issue	Medium	Resolved

ID	Title	Category	Severity	Status
SUA-27	Price Conversion Ignores Feed Decimals Mismatch	Logical Issue	Medium	Resolved
SUA-41	Potential Underflow In `prepareReturn()` Due To Recomputed Total Assets	Incorrect Calculation	Medium	Resolved
SUA-51	The `CrosschainAdapter.setBridgeNonce()` Fails To Call The `BridgeAccountant.forceSetOutboundNonce()`	Access Control	Medium	Resolved
SUA-54	Unclaimable Receiver Can Permanently Lock Redemption Request And Degrade/DoS Subsequent Claims	Logical Issue	Medium	Resolved
SUA-55	Remote Snapshot Reports `unfulfilledWithdrawalAmount` In USDT While Declaring `assetType = USDC`	Volatile Code	Medium	Acknowledged
SUA-03	Missing Check Actual Received Debt Payment	Volatile Code	Minor	Resolved
SUA-04	Missing Check For Feed Configuration	Volatile Code	Minor	Resolved
SUA-08	The `retrieveDebt` Health Check Is Evaluated In The Wrong Context And With Misleading Debt Parameters	Logical Issue	Minor	Acknowledged
SUA-09	Unnecessary `receive()` Function In `OriginVault`	Inconsistency	Minor	Resolved
SUA-12	Outbound Queue DoS Via Duplicate Nonces	Logical Issue	Minor	Acknowledged
SUA-13	Potential Fund Extraction Via `predeposit()` And `instantRedeem()`	Logical Issue	Minor	Acknowledged
SUA-14	Missing Check On `receiver`	Volatile Code	Minor	Resolved
SUA-15	Gas Exhaustion Risk Due To Unbounded Loops In `reconcile()`	Logical Issue	Minor	Resolved

ID	Title	Category	Severity	Status
SUA-16	Missing Revoke Allowance When The External Call Fails	Volatile Code	Minor	Resolved
SUA-28	`SYNC_BRIDGED` Can Be Spoofed Via `sendMessage` Without Funds/Accounting	Logical Issue	Minor	Resolved
SUA-29	External Self-Call Pattern Risks Gas Starvation Of Outer CCIP Receive Frame	Logical Issue	Minor	Resolved
SUA-30	Callback Management Advertised But Not Enforced	Inconsistency	Minor	Acknowledged
SUA-31	Third-Party Dependency Usage	Design Issue	Minor	Acknowledged
SUA-42	Inconsistent Reporting Of Redeemed Shares In `RemoteVault.withdrawFromYearn()`	Inconsistency	Minor	Resolved
SUA-43	The `emergencyWithdraw` Function Does Not Exclude The `usdc` And The `usdo`	Volatile Code	Minor	Acknowledged
SUA-44	Incorrect Event Emission	Logical Issue	Minor	Resolved
SUA-45	Emergency Token Recovery Allows Draining Of Vault Share Balances	Volatile Code	Minor	Resolved
SUA-50	Allowlist Is Only Enforced At Submission Not At Execution	Logical Issue	Minor	Resolved
SUA-53	Changing `bridgeDepositAddress` Can Strand Previously Deposited Funds	Design Issue	Minor	Acknowledged
SUA-56	`defaultGasLimit` Remains Uninitialized	Logical Issue	Minor	Resolved
SUA-06	Function Calls User-Provided Addresses With No Access Control Modifier	Access Control	Informational	Resolved
SUA-10	Missing ERC4626 `Deposit` Events	Code Optimization	Informational	Acknowledged

ID	Title	Category	Severity	Status
SUA-18	Unused Variables	Coding Style	Informational	Resolved
SUA-20	Potential Denial Of Service In Strategy Liquidation Via `liquidateAllPositions()`	Denial of Service	Informational	Acknowledged
SUA-22	Inconsistent Handling Of `premintCooldownVault()` Return Values Across Cooldown Strategies	Logical Issue, Design Issue	Informational	Resolved
SUA-32	Incorrect Comment Implies Optional `SYNC_BRIDGED`	Coding Style	Informational	Resolved
SUA-33	Redundant Statements	Volatile Code	Informational	Resolved
SUA-34	Unused Message Processing Hooks	Volatile Code	Informational	Resolved
SUA-35	Missing Implementation Of Declared Function `deactivateChain()`	Inconsistency	Informational	Resolved
SUA-36	Dead Code	Coding Style	Informational	Resolved
SUA-37	The `USDOKycedCA` Calls An Only-Whitelist Function	Design Issue	Informational	Acknowledged
SUA-39	Discussion On Incomplete Position Adjustment In `StrategyMorphoV1Vault`	Logical Issue	Informational	Acknowledged
SUA-46	Lack Of Reasonable Upper Boundaries	Logical Issue	Informational	Acknowledged
SUA-52	Broken Partial Claim Logic Prevents Redeeming With Slippage	Logical Issue	Informational	Resolved
SUA-57	Permanent Fund Lock Due To Missing Controller Validation	Logical Issue	Informational	Resolved
SUA-58	Incorrect Minimum-Length Check	Logical Issue	Informational	Resolved

ID	Title	Category	Severity	Status
SUA-59	The `redeem()` Does Not Clear Per-Request Amounts And The `claimableRedeemRequest()` Ignores Redeemed Flag	Volatile Code	Informational	Resolved

SUA-47 | Centralized Control Of Contract Upgrade

Category	Severity	Location	Status
Centralization	Centralization	src/superearn/core/CooldownVault.sol (Jan-21): 34 ~35; src/superearn/core/minter/USDOKycedCA.sol (Jan-21): 13~14; src/superearn/v2/base/SuperEarnAccessControl.sol (Jan-21): 46~47; src/superearn/v2/core/crosschain/BridgeAccountant.sol (Jan-21): 56~57; src/superearn/v2/core/crosschain/CrosschainAdapter.sol (Jan-21): 61~62; src/superearn/v2/core/crosschain/SuperEarnMessageAgent.sol (Jan-21): 55~56; src/superearn/v2/core/vaults/OriginVault.sol (Jan-21): 43~44; src/superearn/v2/core/vaults/RemoteVault.sol (Jan-21): 30~31; src/superearn/v2/periphery/AssetPriceConverter.sol (Jan-21): 30 ~31; src/superearn/v2/periphery/OraklAssetPriceConverter.sol (Jan-21): 13~14	Acknowledged

Description

The following contracts are upgradeable and are intended to be used behind a proxy, with upgrades controlled by the proxy admin:

- `CooldownVault`
- `OriginVault`
- `OriginVaultBase`
- `RemoteVault`
- `CrosschainAdapter`
- `SuperEarnAccessControl`
- `BridgeAccountant`
- `SuperEarnMessageAgent`
- `AssetPriceConverter`
- `OraklAssetPriceConverter`
- `USDOKycedCA`
- `RunespearReceiver`
- `RunespearSender`
- `RunespearTransceiver`

If this proxy admin is compromised, an attacker could take advantage of this authority and change the implementation

contract pointed to by the proxy, enabling malicious functionality to be executed through the proxy.

Recommendation

We recommend that the team make efforts to restrict access to the admin of the proxy contract. A strategy of combining a time-lock and a multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to migrate to a new implementation contract.

Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently fully resolve the risk.

Short Term:

A combination of a time-lock and a multi signature (⅔, ⅗) wallet mitigate the risk by delaying the sensitive operation and avoiding a single point of key management failure.

- A time-lock with reasonable latency, such as 48 hours, for awareness of privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromised;
  AND
- A medium/blog link for sharing the time-lock contract and multi-signers addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Long Term:

A combination of a time-lock on the contract upgrade operation and a DAO for controlling the upgrade operation mitigate the contract upgrade risk by applying transparency and decentralization.

- A time-lock with reasonable latency, such as 48 hours, for community awareness of privileged operations;
  AND
- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the time-lock contract, multi-signers addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.

- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Permanent:

Renouncing ownership of the `admin` account or removing the upgrade functionality can *fully* resolve the risk.

- Renounce the ownership and never claim back the privileged role;
  OR
- Remove the risky functionality.

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[SuperEarn, 02/03/2026]: The team acknowledged the issue and adopted the multisign solution to ensure the private key management process at the current stage. The GOVERNANCE_ROLE role has transferred the ownership to a Gnosis Safe contract with 4/5 signers in the sensitive function signing process.

For transparency, the team provided the following multisig and deployment details:

Governance Multisig (Cold Wallet)

Threshold: 4/5, Ethereum Safe: 0xce6917FF9125fff7Da0e5Da5840989B7F3897f2f, Kaia Safe: 0x694B81Db6E16c75B6B9EF9F1b09aa3FD1F1d5f05.

Grant Role/ownership transfer transaction evidence:

Etherscan tx hash: 0x9dacd75446aaece64f42e497f57d2dcba1acb382cd307c1c6ddb006818c39211

The 5 multisig signer addresses:

EOA: 0x449b1cF9632454D60F1B55125F10B475a45c86c6

EOA: 0xbFA4A3430774376CAB83A81FaE301f7b05d90f34

EOA: 0x9A3cE8ca1372114Ec73Ad537F1C6B9594d9FFBbf

EOA: 0x7c0d7cc8d87e78b8167344c13f06ca9f381e07f8

EOA: 0xd7832ad89cbac201f2dc33e41b666f878e882a4f

ProxyAdmin contracts

Owned by Governance multisig and used for upgrades:

Ethereum 0x5732A7422a8f3631a28Ab9439fe9A872BD39418D,

Kaia 0x9Ef977E4521ca735a870cBFA8bA225F558866B4B.

[CertiK, 02/03/2026]: While this strategy reduces the centralization and key-compromise risk, it does not eliminate it entirely. CertiK recommends periodically reviewing the key management and access control of all signer addresses and role holders.

SUA-48 | Centralization Related Risks

Category	Severity	Location	Status
Centralization	Centralization	src/superearn/core/minter/USDOKycedCA.sol (Jan-29): 265~266, 803, 813~814, 826, 826, 836~837, 845, 860, 873, 903, 913, 924~925, 928~929, 938, 955; src/superearn/core/CooldownVault.sol (Jan-21): 228~229, 236~237, 331~332, 352, 377~378, 407~408, 683~684, 827~828, 851~852, 880~881, 890~891, 904~905, 930~931, 939~940, 959~960, 970~971, 981~982, 994~995, 1001~1002, 1009~1010, 1023~1024, 1038~1039, 1241~1242; src/superearn/core/HealthCheck.sol (Jan-21): 28~29, 36~37, 44~45, 52~53, 60~61, 78~79, 88~89; src/superearn/core/strategy/BaseCooldownStrategy.sol (Jan-21): 332~333, 403~404, 420~421, 431~432, 440~441, 453~454, 474~475, 488~489, 498~499; src/superearn/core/strategy/StrategyOriginVault.sol (Jan-21): 145~146, 153~154, 158~159, 399~400, 423~424, 440~441; src/superearn/v2/core/crosschain/BridgeAccountant.sol (Jan-21): 189~190, 189~190, 205~206, 216~217, 228~229, 243~244, 251~252, 260~261, 274~275, 569~570, 580~581, 589~590, 597~598, 606~607, 619~620, 627~628, 642~643; src/superearn/v2/core/crosschain/CrosschainAdapter.sol (Jan-21): 200~201, 231~232, 237~238, 241~242, 245~246, 250~251, 267~268, 537~538, 596~597, 704~705, 713~714, 734~735, 779~780, 794~795; src/superearn/v2/core/crosschain/SuperEarnMessageAgent.sol (Jan-21): 124~125, 152~153, 200~201, 254~255, 289~290; src/superearn/v2/core/vaults/OriginVault.sol (Jan-21): 235~236, 261~262, 267, 273, 280, 290, 300, 471~472, 490~491, 546~547, 717~718, 717~718, 738~739, 854~855; src/superearn/v2/core/vaults/OriginVaultBase.sol (Jan-21): 20~21; src/superearn/v2/core/vaults/RemoteVault.sol (Jan-21): 171~172, 200~201, 214~215, 224~225, 234~235, 250~251, 260~261, 454~455, 500~501, 543~544, 557~558, 584~585, 595~596, 613~614, 652~653, 705~706, 715~716; src/superearn/v2/periphery/AssetPriceConverter.sol (Jan-21): 176~177, 186~187, 196~197, 206~207; src/superearn/v2/periphery/OraklAssetPriceConverter.sol (Jan-21): 102~103, 112~113, 122~123	Acknowledged

Description

In the contract `OriginVault`, the role `GOVERNANCE_ROLE` has authority over the following functions:

- `pause()`
- `unpause()`
- `setAgent()`
- `setFeedProxy()`
- `setPriceConverter()`
- `whitelistShareholder()`
- `removeShareholderFromWhitelist()`
- `emergencyRecoverToken()`
- `depositToRemote()`
- `processRedemptionQueue()`
- `withdrawFromRemote()`
- `batchFulfillRedemptions()`
- `deposit()`
- `mint()`

If a `GOVERNANCE_ROLE` account is compromised, an attacker an attacker may pause all vault operations, recover all tokens to arbitrary recipients, change configurations, manipulate the whitelist, control capital allocation to remote vault, and directly deposit/mint shares bypassing whitelist restrictions.

In the contract `OriginVault`, the role `MANAGEMENT_ROLE` or `KEEPER_ROLE` has authority over the following functions:

- `depositToRemote()`
- `processRedemptionQueue()`
- `withdrawFromRemote()`
- `batchFulfillRedemptions()`

If a `MANAGEMENT_ROLE` account or `KEEPER_ROLE` account is compromised, an attacker may move funds between origin and remote vaults, manipulate redemption queue processing order and timing, potentially causing disruption of normal vault operations.

In the contract `OriginVault`, the role `_whitelistedshareholder` has authority over the following functions:

- `deposit()`
- `mint()`

If a `_whitelistedshareholder` account is compromised, an attacker may mint unlimited shares, inflating the supply and diluting user holdings.

The `OriginVault` contract inherits from `OwnableUpgradeable` from the OpenZeppelin library. As a result, the contract owner (`_owner`) also has authority over the following functions:

- `transferOwnership()`
- `renounceOwnership()`

In the contract `RemoteVault`, the role `GOVERNANCE_ROLE` has authority over the following functions:

- `setYearnVault()`
- `setPriceFeeds()`
- `setPriceConverter()`
- `setSwapRouter()`
- `setSuperEarnRouter()`
- `setMaxSlippage()`
- `setAgent()`
- `emergencyBridgeAssetsToOrigin()`
- `emergencyRecoverToken()`
- `emergencyCooldownVaultRedeem()`
- `emergencyCooldownVaultClaim()`
- `swapUniswap()`
- `depositToYearn()`
- `swapCurve()`
- `withdrawFromYearn()`
- `fulfillPendingWithdrawals()`
- `emergencyWithdrawFromYearn()`

If a `GOVERNANCE_ROLE` account is compromised, an attacker an attacker may recover all tokens to arbitrary recipients, change configurations of agent, price converter, router, change the Yearn Vault address, control capital allocation to Yearn Vault, execute swaps, manipulate pending redemption fulfillment, withdraw assets from Yearn Vault, and manage depositor whitelist.

In the contract `RemoteVault`, the role `KEEPER_ROLE` has authority over the following functions:

- `swapUniswap()`
- `depositToYearn()`
- `swapCurve()`
- `withdrawFromYearn()`
- `fulfillPendingWithdrawals()`

If a `KEEPER_ROLE` account is compromised, an attacker may execute swaps with potential slippage, control capital allocation to Yearn Vault and manipulate pending redemption fulfillment.

In the contract `RemoteVault`, the role `MANAGEMENT_ROLE` has authority over the following function:

- `swapUniswap()`
- `depositToYearn()`

- `swapCurve()`
- `withdrawFromYearn()`
- `fulfillPendingWithdrawals()`
- `emergencyWithdrawFromYearn()`

If a `MANAGEMENT_ROLE` account is compromised, an attacker may waps with potential slippage, control capital allocation to Yearn Vault, manipulate pending redemption fulfillment and withdraw assets from Yearn Vault.

In the contract `RemoteVault`, the role `SYSTEM_CONTRACT_ROLE` has authority over the following function:

- `handleWithdrawRequest()`

If a `SYSTEM_CONTRACT_ROLE` account is compromised, an attacker may manipulate withdrawal request and affect withdrawal fulfillment, leading to accounting inconsistencies.

The `OriginVaultBase` contract inherits from `OwnableUpgradeable` from the OpenZeppelin library. As a result, the contract owner (`_owner`) also has authority over the following functions:

- `transferOwnership()`
- `renounceOwnership()`

In the contract `CooldownVault`, the role `governance` has authority over the following functions:

- `submitCooldownPeriod()`
- `acceptCooldownPeriod()`
- `submitGovernanceTransfer()`
- `setManagement()`
- `addStrategy()`
- `removeStrategy()`
- `addAuthorizedAddress()`
- `removeAuthorizedAddress()`
- `unpause()`
- `recover()`
- `recoverClaimLoss()`
- `pause()`
- `setMaxLossThresholdBps()`
- `setHealthCheck()`
- `setDoHealthCheck()`

If a `governance` account is compromised, an attacker may change cooldown period affecting all redemption requests, transfer governance to attacker-controlled address, add or remove strategies and authorized addresses, pause/unpause vault operations, recover excess assets, recover claim losses by minting shares to themselves, pause vault operations, modify maximum loss threshold for third-party claims and change or disable health check validation.

In the contract `CooldownVault`, the role `management` has authority over the following functions:

- `pause()`
- `setMaxLossThresholdBps()`
- `setHealthCheck()`
- `setDoHealthCheck()`

If a `management` account is compromised, an attacker may pause vault operations, modify maximum loss threshold for third-party claims, change or disable health check validation, potentially allowing unhealthy debt repayments.

In the contract `CooldownVault`, the role `whitelisted strategy` contracts have authority over the following functions:

- `predeposit()`
- `instantRedeem()`
- `retrieveShortfall()`

If a whitelisted `strategy` account is compromised, an attacker may create predeposit debt without providing assets, instantly redeem shares bypassing cooldown period, and manipulate shortfall repayment, potentially causing accounting inconsistencies and fund loss.

In the contract `CooldownVault`, the role `authorized addresses` have authority over the following functions:

- `depositFor()`
- `withdrawTo()`
- `deposit()`
- `mint()`
- `withdraw()`
- `redeem()`

If an `authorized address` account is compromised, an attacker may deposit and mint shares on behalf of any account, initiate withdrawals and redemptions, potentially manipulating vault share supply and user balances.

In the contract `HealthCheck`, the role `governance` has authority over the following functions:

- `setGovernance()`
- `setManagement()`
- `setProfitLimitRatio()`
- `setlossLimitRatio()`
- `setStrategyLimits()`
- `removeStrategyLimits()`
- `setCheck()`

If a `governance` account is compromised, an attacker may transfer governance and management to attacker-controlled addresses, modify profit and loss limits for all strategies or specific strategies and set custom health check contracts.

In the contract `HealthCheck`, the role `management` has authority over the following functions:

- `setProfitLimitRatio()`
- `setlossLimitRatio()`
- `setStrategyLimits()`
- `removeStrategyLimits()`
- `setCheck()`

If a `governance` account or `management` account is compromised, an attacker may modify profit and loss limits for all strategies or specific strategies, set custom health check contracts, potentially allowing unexpected harvests to pass validation and causing fund loss.

In the contract `BaseCooldownStrategy`, the role `governance` has authority over the following functions:

- `emergencyRedeem()`
- `emergencyClaim()`
- `emergencyRepayRemainingPredepositDebt()`
- `setShortfallTolerance()`
- `submitExecution()`
- `acceptExecution()`
- `setAllowedTarget()`
- `cancelExecution()`

If a `governance` account is compromised, an attacker may initiate emergency redemptions and claims from external vaults, manually repay shortfall debt, modify shortfall tolerance, submit and execute arbitrary external calls through timelock mechanism, cancel pending timelock executions and manage allowed target whitelist, potentially causing fund loss or unauthorized operations.

In the contract `BaseCooldownStrategy`, the role `strategist` has authority over the following function:

- `cancelExecution()`

If a `strategist` account is compromised, an attacker may cancel pending timelock executions, potentially disrupting governance operations.

In the contract `BaseCooldownStrategy`, the role `CooldownVault` contract has authority over the following function:

- `repayPredepositDebt()`

If the `CooldownVault` contract is compromised, an attacker may trigger debt repayment, potentially manipulating predeposit debt accounting and causing fund loss.

In the contract `StrategyOriginVault`, the role `governance` has authority over the following functions:

- `setCooldownPeriod()`
- `setReserveRatio()`
- `emergencyRedeem()`

- `emergencyClaim()`
- `emergencyClearRedemptions()`

If a `governance` account is compromised, an attacker may modify cooldown period affecting redemption timing, initiate emergency redemptions without predeposit backing, claim redemptions potentially causing double-counting in estimatedTotalAssets, and clear all redemption tracking state for emergency migration, potentially causing accounting inconsistencies and fund loss.

In the contract `StrategyUSDOExpressV2`, the role `governance` has authority over the following function:

- `setReserveRatio()`

If a `governance` account is compromised, an attacker could modify the reserve ratio to manipulate investment shares.

In the contract `BridgeAccountant`, the role `GOVERNANCE_ROLE` has authority over the following functions:

- `setAdapter()`
- `forceManualClearOutboundByNonce()`
- `forceManualClearOutboundByAmount()`
- `clearExpiredOutboundAwaitingPeerReceipt()`
- `forceManualReleaseInboundByNonce()`
- `forceManualReleaseInboundByAmount()`
- `clearExpiredInboundAwaitingPeerRelease()`

If a `GOVERNANCE_ROLE` account is compromised, an attacker may change the adapter address, manipulate outbound and inbound nonces, clear bridge operations and modify bridge timeout duration, potentially causing accounting inconsistencies.

In the contract `BridgeAccountant`, the role `MANAGEMENT_ROLE` has authority over the following function:

- `setBridgeTimeout()`

If a `MANAGEMENT_ROLE` account is compromised, an attacker may modify bridge timeout duration, potentially affecting bridge operations.

In the contract `BridgeAccountant`, the `adapter` contract has authority over the following functions:

- `allocateOutboundNonce()`
- `recordOutbound()`
- `recordInbound()`
- `addAwaitingAssetNotification()`
- `removeAwaitingAssetNotification()`
- `updatePeerSnapshot()`
- `reconcileBridgeState()`
- `forceSetOutboundNonce()`

If the `adapter` contract is compromised, an attacker may allocate and record bridge operations, manipulate peer snapshot data, reconcile bridge state incorrectly, potentially causing accounting inconsistencies and incorrect totalAssets calculations, causing incorrect vault share pricing.

In the contract `CrosschainAdapter`, the role `GOVERNANCE_ROLE` has authority over the following functions:

- `setLocalVaultRole()`
- `setBridgeDepositAddress()`
- `setVault()`
- `setBridgeNonce()`
- `setAssetTypeToken()`
- `setAgent()`
- `configurePeer()`
- `forceProcessBridgeReceipt()`
- `sweepToken()`
- `sweepEth()`
- `removeFailedMessage()`

If a `GOVERNANCE_ROLE` account is compromised, an attacker may remove failed messages, change vault and agent addresses, configure peer chain settings, manipulate bridge nonces, force process bridge receipts, sweep tokens, potentially causing cross-chain communication disruption and fund loss.

In the contract `CrosschainAdapter`, the role `MANAGEMENT_ROLE` has authority over the following function:

- `removeFailedMessage()`

If a `MANAGEMENT_ROLE` account is compromised, an attacker may permanently remove failed messages from storage, causing cross-chain state inconsistencies.

In the contract `CrosschainAdapter`, the role `GOVERNANCE_ROLE`, `MANAGEMENT_ROLE`, or `KEEPER_ROLE` has authority over the following functions:

- `processPendingBridgeAssets()`
- `retryFailedMessage()`
- `sendMessage()`

If a `vault`, `agent`, `GOVERNANCE_ROLE`, `MANAGEMENT_ROLE`, or `KEEPER_ROLE` account is compromised, an attacker may process pending bridge assets, retry failed messages, and send cross-chain messages, potentially causing bridge state updated incorrectly and cross-chain communication disruption.

In the contract `SuperEarnMessageAgent`, the role `GOVERNANCE_ROLE` has authority over the following function:

- `setAdapter()`

If a `GOVERNANCE_ROLE` account is compromised, an attacker may change the adapter address, causing communication failures between chains.

In the contract `SuperEarnMessageAgent`, the `adapter` contract has authority over the following functions:

- `delegate()`
- `sendBridgedAssets()`

If the `adapter` contract address is compromised, an attacker may route cross-chain messages incorrectly, transfer bridged assets to arbitrary vault addresses, causing cross-chain state inconsistencies and fund loss.

In the contract `SuperEarnMessageAgent`, the `vault` contract has authority over the following functions:

- `sendMessage()`
- `prepareAndSendAssets()`

If the `vault` contract is compromised, an attacker may send cross-chain messages and pull assets from vault for transfer, causing message spam and fund loss.

In the contract `AssetPriceConverter`, the role `GOVERNANCE_ROLE` has authority over the following functions:

- `setConstantDecimals()`
- `setMaxPriceAge()`
- `setMinStablecoinPrice()`
- `setMaxStablecoinPrice()`

If a `GOVERNANCE_ROLE` account is compromised, an attacker may modify price conversion parameters, change staleness checks, and adjust stablecoin price bounds, causing incorrect asset configurations and affecting vault totalAssets calculations and share pricing.

In the contract `OraklAssetPriceConverter`, the role `GOVERNANCE_ROLE` has authority over the following functions:

- `setMaxPriceAge()`
- `setMinStablecoinPrice()`
- `setMaxStablecoinPrice()`

If a `GOVERNANCE_ROLE` account is compromised, an attacker may modify price checks and stablecoin price bounds, causing incorrect totalAssets calculations and share pricing.

In the contract `USDOKycedCA`, the role `governance` has authority over the following functions:

- `setCooldownPeriod()`
- `setHealthCheck()`
- `setDoHealthCheck()`
- `setHooks()`
- `addStrategy()`
- `removeStrategy()`
- `syncHistoricalMinRedeemAmt()`
- `submitGovernanceTransfer()`

- `pause()`
- `unpause()`
- `recover()`
- `emergencyWithdraw()`
- `claim()`

If a `governance` account is compromised, an attacker may pause all operations, change cooldown periods, modify health check configurations, set malicious hooks contract, add or remove whitelisted strategies, recalculate historical minimum redeem amount, transfer governance to attacker-controlled address, withdraw any tokens, and claim redeem request, leading to accounting inconsistencies and fund loss.

In the contract `USDOKycedCA`, the role `pendingGovernance` has authority over the following functions:

- `acceptGovernanceTransfer()`

If a `pendingGovernance` account is compromised, an attacker may accept governance transfer and cause the fund loss.

In the contract `USDOKycedCA`, the role `whitelisted strategy` contracts have authority over the following functions:

- `deposit()`
- `redeem()`
- `claim()`

If a whitelisted `strategy` account is compromised, an attacker may redeem USDO for USDC and claim USDC, causing accounting inconsistencies and fund loss.

In the contract `USDOKycedCA`, the `hooks` contract has authority over the following functions:

- `deposit()`
- `redeem()`
- `claim()`

If the `hooks` contract is compromised, an attacker may execute arbitrary code during deposit, redeem, and claim operations, leading to accounting inconsistencies and fund loss.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[SuperEarn, 02/03/2026]: The team acknowledged the issue and adopted the multisign solution to ensure the private key management process at the current stage. The project has transferred ownership/admin control to a Gnosis Safe contract, with the Governance multisig responsible for GOVERNANCE_ROLE and the Management multisig responsible for MANAGEMENT_ROLE. The GOVERNANCE_ROLE role has transferred the ownership to a Gnosis Safe contract with 4/5 signers. The MANAGEMENT_ROLE role has transferred the ownership to a Gnosis Safe contract with 2/3 signers.

For transparency, the team provided the following multisig and deployment details:

Governance Multisig (Cold Wallet)

Threshold: 4/5, Ethereum Safe: 0xce6917FF9125fff7Da0e5Da5840989B7F3897f2f, Kaia Safe: 0x694B81Db6E16c75B6B9EF9F1b09aa3FD1F1d5f05.

Grant Role/ownership transfer transaction evidence:

Etherscan tx hash: 0x9dacd75446aaece64f42e497f57d2dcba1acb382cd307c1c6ddb006818c39211

The 5 multisig signer addresses:

EOA: 0x449b1cF9632454D60F1B55125F10B475a45c86c6

EOA: 0xbFA4A3430774376CAB83A81FaE301f7b05d90f34

EOA: 0x9A3cE8ca1372114Ec73Ad537F1C6B9594d9FFBbf

EOA: 0x7c0d7cc8d87e78b8167344c13f06ca9f381e07f8

EOA: 0xd7832ad89cbac201f2dc33e41b666f878e882a4f

Management Multisig (Hot Wallet)

Threshold: 2/3, Ethereum Safe: 0xd85Ba41BFe1519813224Be0ba87974cADf3AD3A0, Kaia Safe: 0xAf65b84B306b4a39EcCc3c915A9143956BC935C8.

The 3 multisig signer addresses:

EOA: 0x9fa3af580e4d27bf11f49f6c582d3dc40140d6a4

EOA: 0x374f7713ab5a50ce7e779bbf71b803f311352bf3

EOA: 0x19c7cf8c28df30963b6436a22d4eee0cd99585f7

[CertiK, 02/03/2026]: While this strategy reduces the centralization and key-compromise risk, it does not eliminate it entirely. CertiK recommends periodically reviewing the key management and access control of all signer addresses and role holders. Additionally, the project should confirm and document the production setup for any remaining privileged roles (e.g., KEEPER_ROLE) before the finding can be considered fully mitigated.

SUA-11 Fixed Request-Time `requestedAssets` Can Permanently Deadlock The Redemption Queue After Losses/Shortfalls

Category	Severity	Location	Status
Logical Issue	Major	src/superearn/v2/core/vaults/OriginVault.sol (Dec-19): 544~545, 613	Acknowledged

Description

The amount paid on redemption is effectively fixed at `requestRedeem()` time, and `batchFulfillRedemptions()` requires fulfilling each queue item atomically for exactly that amount (sequentially, no skipping, no partial fulfillment).

```
 598   function requestRedeem(
 599           uint256 shares,
 600           address controller,
 601           address owner
 602       )
 603           external
 604           override
 605           whenNotPaused
 606           returns (uint256 requestId)
 607       {
 608           if (owner != msg.sender && !isOperator[owner][msg.sender]) revert
InvalidOwner();
 609           if (balanceOf(owner) < shares) revert InsufficientBalance();
 610           if (shares == 0) revert ZeroShares();
 611
 612
// Lock the expected asset amount at request time to avoid NAV drift between request
and processing.

 613  @>       uint256 requestedAssets = convertToAssets(shares);
 614           ...
 615   }
```

If the vault's realizable assets-per-share decreases between request and fulfillment (e.g., remote vault loss, negative yield, bridge shortfall/fees, or any scenario where the maximum assets the vault can obtain becomes less than the earliest item's `requestedAssets`), then the first such item becomes impossible to fulfill and blocks all later redemptions indefinitely.

```
 544    function batchFulfillRedemptions(uint256 maxAmountUsdt, uint256 maxCount)
external whenNotPaused onlyOperators {
 545         ...
 546         uint256 startIndex = queueFulfilledIndex;
 547         uint256 endIndex =
 548             queueRemoteRequestedIndex < redemptionQueue.length ?
 queueRemoteRequestedIndex : redemptionQueue.length;
 549         endIndex = endIndex - startIndex < maxCount ? endIndex : startIndex +
 maxCount;
 550
 551         uint256 availableAssets = fulfillmentEligibleAssets();
 552         uint256 maxToFulfill = maxAmountUsdt > availableAssets ?
 availableAssets : maxAmountUsdt;
 553
 554
// Process items sequentially from queueFulfilledIndex up to
queueRemoteRequestedIndex

 555
// Items before queueRemoteRequestedIndex have been requested, items after haven't
 556         // Each item must be processed atomically, with no partial fulfillment
 557         for (uint256 i = startIndex; i < endIndex; i++) {
 558             RedemptionQueueItem storage item = redemptionQueue[i];
 559
 560  @>     uint256 reservedAmount = item.requestedAssets;
 561         ...
 562         }
 563  }
```

We would like to raise this as a finding and discuss with the team whether this scenario and the associated risks have been considered. If not, we recommend representing queued redemptions in shares rather than assets. Specifically, the requester's shares should be locked and recorded at the time of queuing, with the final payout calculated at fulfillment using `convertToAssets(shares)` based on the then-current exchange rate.

Recommendation

Consider representing queued redemptions in shares, not assets. Lock and store the requester's shares at queue time and compute payout using current `convertToAssets(shares)` at fulfillment.

Alleviation

[SuperEarn, 01/12/2026]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement.

During the process of transferring assets from a remote vault, losses may occur. Conversely, even if a withdrawal request is made from the origin vault to a remote vault, the withdrawal is not executed immediately and is subject to a delay. Because of this delay, swap rates may change, and during the process of swapping strategy assets into USDT, the outcome may differ from the request-time state-resulting in gains, additional interest accrual, or losses. Accordingly, there are many factors that can cause the final amount to deviate from what was observed at the time the request was submitted.

At present, any losses or profits arising from these processes are socialized among the remaining users. For operational costs such as bridge fees, we use a partner bridge under a commercial agreement, and the protocol covers these fees on behalf of users. From an operational standpoint, we aim to minimize user-facing costs by absorbing as much of the deposit and withdrawal-related fees as possible at the protocol level.

Given that withdrawals are not instantaneous and involve a delay, users may experience a discrepancy between the amount shown at the time of the withdrawal request and the amount actually received. Between the two UX trade-offs-(i) exposing users to request-time versus execution-time amount discrepancies, and (ii) socializing the gains and losses generated during the deposit and withdrawal processes among the remaining users-we have deliberately chosen the latter in order to provide a more consistent and predictable user experience.

SUA-01 | Potential Share Loss When Claiming

Category	Severity	Location	Status
Volatile Code	Medium	src/superearn/core/CooldownVault.sol (Dec-19): 407, 514, 605, 1246	Acknowledged

Description

The `CooldownVault` contract implements a two-step redemption process where users first request to redeem shares with the `redeem` function, and then claim the corresponding assets through the `claim` function, which calls the internal `_claim` function.

```
605         totalClaimLoss += request.assets - assetsOut;
```

However, when the accumulated claim loss is later recovered using the `recoverClaimLoss` function, the contract mints the corresponding amount of shares to the governance address, not to the users who experienced the loss:

```
1246     function recoverClaimLoss() external
override nonReentrant onlyGovernance returns (uint256 assets) {
1247         assets = totalClaimLoss;
1248         _mint(_msgSender(), assets); // equals shares minted to governance
1249         totalClaimLoss = 0;
1250
1251         emit RecoverClaimLoss(_msgSender(), assets);
1252     }
```

As a result, users who incur claim losses do not receive compensation for the lost assets, and only the governance account benefits from the recovery process. This design can lead to user dissatisfaction.

Recommendation

It's recommended that user compensation is prioritized by ensuring that any shares or assets recovered after a claim loss are distributed back to the users who experienced the loss, rather than being allocated to the governance address.

Alleviation

[SuperEarn, 01/08/2026]: The team acknowledged the issue and decided not to make code changes at this stage.

SUA-02 | Unclaimable Redemptions From Zero Asset Amounts

Category	Severity	Location	Status
Volatile Code	Medium	src/superearn/v2/core/vaults/OriginVault.sol (Dec-19): 613	Resolved

Description

`requestRedeem()` queues a redemption by transferring the user's shares into the vault. The request can later be fulfilled and then redeemed for assets.

However, when `totalAssets()` is very small, `convertToAssets(shares)` may round down to 0. This allows redemption requests with `requestedAssets == 0` to be queued. The `redeem()` function will revert on ZeroAssets(), leaving the shares stuck in the vault and unclaimable.

Scenario

For example the underlying asset is USDC:

1. deposit 1 wei USDC, `_convertToShares` gives 10**12 shares (_decimalsOffset=12);
2. totalSupply ≈ 1e12 shares and totalAssets ≈ 1 wei USDC;
3. redeems 1 wei of those shares, `_convertToAssets` computes ≈ 1 * 2 / (1e12 + 1e12), which floors to 0.

That `requestRedeem()` records `requestedAssets=0`; when fulfilled, `redeem()` reverts on ZeroAssets(), so that 1 wei share gets stuck in the vault.

Recommendation

Add a zero check on requestedAssets:

```
614  if (requestedAssets == 0) revert ZeroAssets();
```

Alleviation

[SuperEarn, 01/04/2026]: The team heeded the advice and resolved the issue by adding a zero check in commit 21548432899a5cab783335a2cfaf966782af9c40.

SUA-21 | Over-Permissive Access To `sendAssets`

Category	Severity	Location	Status
Access Control	Medium	src/superearn/v2/core/crosschain/CrosschainAdapter.sol (Dec-19): 364~366	Resolved

Description

The function `sendAssets()` allows calls from both vault and agent. In practice, only `SuperEarnMessageAgent.prepareAndSendAssets()` uses it. `OriginVault` / `RemoteVault` never call it. This permissiveness breaks the stated design of keeping vaults crosschain-agnostic and routed via the agent.

Recommendation

Enforce the `msg.sender` to be agent.

Alleviation

[SuperEarn, 01/20/2026]: The team heeded the advice and resolved the issue in commit b7ae75a354f26bd16c185ffb6042f7d24b91de31.

SUA-23 Incorrect Supply Cap Calculation Causes Underutilization Of Investment Capacity

Category	Severity	Location	Status
Logical Issue	Medium	src/superearn/core/BaseCooldownStrategy.sol (Dec-19): 212; src/superearn/core/StrategyUSDOExpressV2.sol (Dec-19): 179, 299	Resolved

Description

The `StrategyUSDOExpressV2` strategy includes an internal constraint `MAX_SUPPLY_THRESHOLD` to limit the total amount of USDT that can be deposited. The `getSupplyCap()` function is designed to return both the protocol's maximum deposit capacity and the actual depositable amount after applying internal constraints.

The `adjustPosition()` function calls `getSupplyCap()` to determine the available amount before reaching the cap. The root cause is in the `getSupplyCap()` implementation:

```
// src/superearn/core/StrategyUSDOExpressV2.sol:179
function getSupplyCap() public view virtual override returns (uint256
supplyAssetsCap, uint256 availableAssets) {
    supplyAssetsCap = externalUnderlyingToken.balanceOf(address(this)) + oneUsdt;
    availableAssets = Math.min(supplyAssetsCap, MAX_SUPPLY_THRESHOLD);
}
```

The function sets `availableAssets` to the minimum of the current USDT balance (plus a 1 USDT buffer) and `MAX_SUPPLY_THRESHOLD`. This is incorrect because `availableAssets` should represent "The amount still available to be supplied before reaching the cap" (`BaseCooldownStrategy.sol:212`).

As a result, when `adjustPosition()` uses this `_availableUsdt` value to cap `toInvestShares` via `cooldownVault.previewDeposit(_availableUsdt)`, the strategy effectively limits new investments to its existing USDT balance rather than the true remaining capacity. In practice, this restricts investments to around 1 USDT (or causes a revert due to minimum deposit constraints).

Consequently, the strategy fails to deploy available funds that could otherwise generate yield, leaving capital idle and resulting in suboptimal returns for vault depositors.

Recommendation

It's recommended that the `getSupplyCap()` function be corrected to accurately compute the remaining capacity available before reaching `MAX_SUPPLY_THRESHOLD`, rather than relying on the contract's current token balance. Specifically, the calculation should determine how much more can be invested without exceeding the supply cap, so that the investment logic utilizes the full available allocation, thereby maximizing yield for depositors and preventing underutilization of idle funds.

Alleviation

[SuperEarn, 01/18/2026]: In commit 4d3d345b8ceefdd00de3298baa03d77a4320937c, the getSupplyCap() function was updated to use the maximum value as supplyAssetsCap.

SUA-24 | Bridge Reconciliation Runs On Stale Snapshots

Category	Severity	Location	Status
Logical Issue	Medium	src/superearn/v2/core/crosschain/BridgeAccountant.sol (Dec-19): 261, 276; src/superearn/v2/core/crosschain/CrosschainAdapter.sol (Dec-19): 671~672; src/superearn/v2/messaging/ccip/CCIPReceiverUpgradeable.sol (Dec-19): 39~44	Resolved

Description

The function `_handle()` will be triggered by the `onlyRouter` function `CCIPReceiverUpgradeable.ccipReceive()`. CCIP's `onlyRouter` only guarantees the message came through the configured CCIP router. Combined with `RunespearReceiver` checks, it cannot guarantee the payload's freshness.

The `_handle()` treats the `peerSnapshot.vaultState.timestamp` timestamp as a freshness gate for `peerSnapshot`, but does not apply the same gate to bridge reconciliation.

```
666     function _handle(uint256 sourceChainId, bytes4 predicate, bytes memory args
, bytes32 messageId) internal override {
667         RunespearProtocol.RunespearMessageEnvelope memory envelope =
668             abi.decode(args, (RunespearProtocol.RunespearMessageEnvelope));
669
670         // ALWAYS update peer snapshot and reconcile (on EVERY message)
671  @>     _accountant.updatePeerSnapshot(envelope.stateSnapshot);
672  @>     _accountant.reconcileBridgeState(sourceChainId, envelope.stateSnapshot.
bridgeState);
673         ...
674 }
```

```
260     function updatePeerSnapshot(SuperEarnV2Protocol.StateSnapshot memory
 snapshot) external override onlyAdapter {
261  @>       if (snapshot.vaultState.timestamp >= peerSnapshot.vaultState.
timestamp) {
262             peerSnapshot = snapshot;
263             emit PeerSnapshotUpdated(snapshot.vaultState.timestamp, snapshot.
vaultState.totalAssets);
264         }
265     }
```

This means a message carrying an *older* snapshot (one that is rejected by `updatePeerSnapshot()`) can still feed an outdated `bridgeState` into `BridgeQueue.reconcile()` and mutate `_outboundTracker/_inboundTracker`.

```
276 function reconcileBridgeState(
277         uint256 sourceChainId,
278         RunespearProtocol.BridgeState memory peerBridgeState
279     )
280         external
281         override
282         onlyAdapter
283         returns (BridgeQueue.ReconciliationResult memory result)
284     {
285         result = BridgeQueue.reconcile(
286             _outboundTracker,
287             _inboundTracker,
288             peerBridgeState.inboundAwaitingPeerRelease,
289             peerBridgeState.outboundAwaitingPeerReceipt
290         );
291         ...
292 }
```

Recommendation

Gate bridge reconciliation behind the same freshness check used for peer snapshots. If an incoming `StateSnapshot` is older than the stored peer snapshot, skip reconciliation entirely.

Alleviation

[SuperEarn, 01/19/2026]: The team heeded the advice and resolved the issue in commit 3b1c783f21ee8f790e1440edcdeb3ce916dab73a.

SUA-25 | Incomplete Round-Based Protection

Category	Severity	Location	Status
Logical Issue	Medium	src/superearn/v2/messaging/runespear/RunespearLib.sol (Dec-19): 231~244; src/superearn/v2/messaging/runespear/RunespearReceiver.sol (Dec-19): 185~188; src/superearn/v2/messaging/runespear/RunespearTransceiver.sol (Dec-19): 53~63	Acknowledged

Description

Despite the claim of round-based protection, when a message arrives with zero remaining rounds the code merely notes it and emits `MaxRoundsReached`, then still processes the message, meaning the “no more rounds” condition is ignored instead of stopping the exchange.

```
184          // Check if we've reached the max rounds limit
185          if (!RunespearLib.canContinueRound(runespearMessage)) {
186              emit MaxRoundsReached(sourceChainId, message.messageId,
 runespearMessage.predicate, runespearMessage.args);
187  @>
// Still process the message, but it won't be able to generate further responses
188          }
189
190          // Process the message
191  @>          _processRunespearMessage(sourceChainId, runespearMessage, message.
messageId);
```

Although the function `validateRounds` can revert when no more rounds and the `createResponseMessage` does decrement `remainingRounds`, but both are not called on the receive path.

```
231      function canContinueRound(Message memory message) internal pure returns (
bool) {
232          return message.header.remainingRounds > 0;
233      }
234
235      /**
236       * @notice Validate that the message has remaining rounds
237       * @param message The message to validate
238       * @dev Reverts with NoRemainingRounds if validation fails
239       */
240      function validateRounds(Message memory message) internal pure {
241          if (message.header.remainingRounds == 0) {
242              revert NoRemainingRounds();
243          }
244      }
```

```
     function createResponseMessage(
        Message memory originalMessage,
        bytes4 responsePredicate,
        bytes memory responseArgs
    )
        internal
        pure
        returns (bytes memory)
    {
        // Validate we can still respond
        validateRounds(originalMessage);

        // Decrement the remaining rounds
@>      uint8 nextRemainingRounds = originalMessage.header.remainingRounds - 1;

        return encodeMessageWithRounds(
            responsePredicate, responseArgs,
originalMessage.header.callbackHintHash, nextRemainingRounds
        );
    }
```

Round limits are ineffective, messages can loop indefinitely or be retried without bound, undermining the advertised safety control and potentially enabling repeated re-entry/processing of stale messages.

Recommendation

Call `RunespearLib.validateRounds()` inside `RunespearReceiver.processMessage()` to reject messages whose remaining rounds are 0.

Use `createResponseMessage` to send response.

Alleviation

[SuperEarn, 01/20/2026]: The team called `validateRounds()` instead of emitting the event in commit 6a6e01182aa46c904b9ff912722ff5380f42d32b.

The team later removed this feature in commit 10f58de41e6ccdcfe70e1e810298793113891ca6.

Moreover, in this version, all messages are sent as VOID_CALLBACK, and there is no code path to send a response. As a result, remainingRounds is never decremented for any message, and response-sending logic is not implemented.

SUA-26 Incomplete Liquidation Due To Ignored Underlying Token Balance

Category	Severity	Location	Status
Logical Issue	Medium	src/superearn/core/BaseCooldownStrategy.sol (Dec-19): 278~279	Resolved

Description

The `liquidateAllPositions()` function fails to fully liquidate `externalUnderlyingToken` (e.g., USDT) that is held directly by the strategy. This results in an incomplete liquidation during strategy exit and may lead to fund loss.

Specifically:

1. `liquidateAllPositions()` calls `premintCooldownVault(estimatedTotalAssets() + oneUsdt)`, which computes the strategy's total assets, including both sUSDT and USDT.
2. Within `premintCooldownVault()`, the number of sUSDT shares to redeem is capped via `Math.min`, causing the strategy to redeem all available sUSDT.
3. The function then calls `cooldownVault.deposit(redeemUsdt, ...)` (or `predeposit`), but only deposits the `redeemUsdt` obtained from the redemption.
4. Regardless of whether the USDT comes from redemption or was already held by the strategy prior to the call, the function only deposits the redeemed amount (`redeemUsdt`) into the `cooldownVault` and completely ignores the locally held USDT, resulting in an incomplete liquidation.

As a result, the pre-existing USDT balance is never converted into `want` tokens. Since `_amountFreed` only reflects the `want` balance, it is lower than the true redeemable value.

```
    function liquidateAllPositions() internal virtual override nonReentrant returns
(uint256 _amountFreed) {
        premintCooldownVault(estimatedTotalAssets() + oneUsdt);
        _amountFreed = want.balanceOf(address(this));
    }
```

Recommendation

When `_needSusdt > _susdtBalance` or `remainingPredepositDebt > 0`, the strategy should first use the existing underlying assets to repay the shortfall (`repayRemainingPredepositDebt`), then deposit any remaining underlying assets into the `cooldownVault` before proceeding with the rest of the flow.

Alleviation

[SuperEarn, 01/26/2026]: The team heeded the advice and resolved the issue by adding a `_redepositUnderlyingSurplus()` function to deposit all surplus USDT in commit

86c7d43becf7a901637cb4dbdbe12b4f6440d535.

SUA-27 | Price Conversion Ignores Feed Decimals Mismatch

Category	Severity	Location	Status
Logical Issue	Medium	src/superearn/v2/periphery/AssetPriceConverter.sol (Dec-19): 148~150	Resolved

Description

The function `convertTokenAmount()` uses raw prices from `getLatestPrice()` without normalizing feed decimals. The `getLatestPrice()` reads `feed.decimals()` but neither enforces `decimals == constantDecimals` nor rescales the answer. As a result, `tokenInAmount` (token decimals) multiplied by `tokenInPrice` (feed decimals) is later divided by `tokenOutPrice` that may be on a different scale. If the two feeds use different decimals(which is plausible across oracle providers/networks), the conversion is off by up to 10^(decimalsDiff), mispricing swaps.

```
148     tokenOutAmount =
149             (tokenInAmount * tokenInPrice * (10 ** tokenOutDecimals)) /
tokenOutPrice / (10 ** tokenInDecimals);
150
```

In `RemoteVault`, these conversions feed into `totalAssets()` and related accounting, which can misprice shares and break core vault invariants.

Recommendation

To avoid this, normalize prices to a single basis (or require matching feed decimals) before doing `amount * price_in / price_out`.

Alleviation

[SuperEarn, 01/20/2026]: The team heeded the advice and resolved the issue by adding price normalization in commit 822f0fe571eeee37b96ed6d42ccfd112e4eb4408.

SUA-41 Potential Underflow In `prepareReturn()` Due To Recomputed Total Assets

Category	Severity	Location	Status
Incorrect Calculation	Medium	src/superearn/core/strategy/StrategyUSDOExpressV2.sol (Jan-21): 223~224	Resolved

Description

The `prepareReturn()` function computes the profit and debt repayment for the `USDOKycedCA` vault.

It first determines `_debtPayment` as the minimum of `_debtOutstanding` and the current `totalAssets`, where `totalAssets` is obtained from `estimatedTotalAssets()` before liquidation:

```
        // 1. Calculate current total assets
        uint256 totalAssets = estimatedTotalAssets();
        uint256 totalDebt = getStrategyParams().totalDebt;

        // 2. Estimate debt repayment and profit/loss
        _debtPayment = Math.min(_debtOutstanding, totalAssets);
```

After the `liquidatePosition()` operation, the function recomputes `totalAssets` by calling `estimatedTotalAssets()` again and then subtracts `_debtPayment`. However, at this point `_debtPayment` is only capped by `toReturn`, not by the newly recomputed `totalAssets`:

```
        // 3. Actual repayment: Since the expected and actual repayment amounts may
differ during liquidation,
        // recalculate accurately based on the realizable amount

        // yVault enforces: want.balanceOf(address(this)) >= _profit + _debtPayment
        (uint256 toReturn,) = liquidatePosition(_profit + _debtPayment);

        totalAssets = estimatedTotalAssets();
        totalDebt = getStrategyParams().totalDebt;

        _debtPayment = Math.min(_debtPayment, toReturn);
        totalAssets -= _debtPayment;
        totalDebt -= _debtPayment;
```

If `estimatedTotalAssets()` decreases after `liquidatePosition()`, `totalAssets` may become smaller than `_debtPayment`. This can occur, for example, when `remainingPredepositDebt` becomes greater than or equal to the post-liquidation `pendingInvested + previewInWant`, causing `estimatedTotalAssets()` to return zero.

In such cases, the operation `totalAssets -= _debtPayment` will underflow and revert. This behavior can cause `harvest()` to revert, resulting in a denial-of-service condition under realistic shortfall scenarios.

Recommendation

It is recommended to refactor this function to ensure that no underflow or unexpected revert can occur.

Alleviation

[SuperEarn, 01/26/2026]: The team heeded the advice and resolved the issue by adding the following code snippet in commit 2f1123ab2a4c73175a338eb16f1305d70805bc4b:

```
totalAssets = totalAssets > _debtPayment ? totalAssets - _debtPayment : 0;
```

SUA-51 The `CrosschainAdapter.setBridgeNonce()` Fails To Call The `BridgeAccountant.forceSetOutboundNonce()`

Category	Severity	Location	Status
Access Control	Medium	src/superearn/v2/core/crosschain/CrosschainAdapter.sol (Feb-2): 241~243	Resolved

Description

The `CrosschainAdapter.setBridgeNonce()` function is restricted to the governance role and is designed to update the bridge nonce by invoking `_accountant.forceSetOutboundNonce(newNonce)`.

However, the `BridgeAccountant.forceSetOutboundNonce()` function is also restricted to calls from the governance address. When `setBridgeNonce()` is invoked, the call to `forceSetOutboundNonce()` originates from the `CrosschainAdapter` contract rather than directly from the governance address.

As a result, unless the `CrosschainAdapter` is explicitly granted the `GOVERNANCE_ROLE` within the `BridgeAccountant` contract, the access control check in `forceSetOutboundNonce()` will fail and the call will revert.

Notably, the `BridgeAccountant` contract already defines an `onlyAdapter` modifier to restrict functions to calls originating from the `CrosschainAdapter`. We recommend confirming with the team whether this dual access control design is intentional, or whether the potential risk of disabling nonce recovery via the adapter has been fully considered.

Recommendation

Either elevate `CrosschainAdapter` to a permitted caller on `BridgeAccountant.forceSetOutboundNonce()` or expose a dedicated `forceSetOutboundNonceFromAdapter` (onlyAdapter) that the adapter can safely call.

Alleviation

[SuperEarn, 02/09/2026]: The team heeded the advice and resolved the issue by applying `onlyAdapter` modifier to `BridgeAccountant.forceSetOutboundNonce()` in commit bf3c3c42b7343496cb570ebd695f358807823724.

SUA-54 Unclaimable Receiver Can Permanently Lock Redemption Request And Degrade/DoS Subsequent Claims

Category	Severity	Location	Status
Logical Issue	Medium	src/superearn/core/CooldownVault.sol (Feb-2): 613	Resolved

Description

In SuperEarnRouter, users can deposit underlying tokens into a Yearn vault for a specific recipient.

```
  function deposit(
        address yVault,
        uint256 amount,
@>      address receiver,
        uint256 minSharesOut
    )
        external
        returns (uint256)
    {
        address sender = msg.sender;
        if (remoteVault != address(0) && sender != remoteVault) {
            revert Unauthorized();
        }

        return _deposit(yVault, amount, receiver, minSharesOut);
    }
```

When claiming, the `_claim()` function unconditionally transfers underlying tokens to the stored `request.receiver` via `safeTransfer()`.

According to the document, the underlying token is USDC/USDT with blacklist functionality. If the receiver set in the deposit time is blacklisted, the transfer will always revert and the request can never be marked `claimed`.

This leaves `totalLockedAssets` and the request's amount effectively permanently reserved, which can reduce available liquidity for other requests and can block later claims via the `reservedForPriorRequests` check. There is no mechanism to change the receiver so funds can become stuck indefinitely once a receiver becomes non-payable.

Recommendation

It's recommended that the contract logic allows the receiver address on a pending redemption request to be updated in scenarios where the current receiver is rendered unclaimable (such as being blacklisted by the token contract), provided appropriate authorization checks are implemented.

Alleviation

[SuperEarn, 02/09/2026]: The team heeded the advice and resolved the issue by adding a new function `updateRedeemReceiver` in commit 0d0f1e85f24096f6dcb5370b896a25b673ee0132.

SUA-55 Remote Snapshot Reports `unfulfilledWithdrawalAmount` In USDT While Declaring `assetType = USDC`

Category	Severity	Location	Status
Volatile Code	Medium	src/superearn/v2/libraries/VaultStateHelper.sol (Feb-2): 62~63	Acknowledged

Description

The `getRemoteState()` returns a `SuperEarnV2Protocol.VaultState` snapshot with assetType hardcoded to `SuperEarnV2Protocol.AssetType.USDC` and populates `totalAssets/idleAssets` using `RemoteVault` accounting expressed in USDC terms.

However, it also sets `unfulfilledWithdrawalAmount` from `IRemoteVault(vault).getUnfulfilledWithdrawalInfo()`, which per the `IRemoteVault` documentation and `RemoteVault.unfulfilledWithdrawalAmount` usage is tracked/returned in USDT terms. This breaks the documented `SuperEarnV2Protocol.VaultState` invariant that all amounts in the snapshot must be denominated in the currency specified by `assetType`.

As a result, any consumer of the snapshot (off-chain automation/monitoring today and potentially on-chain logic in future upgrades) may misinterpret unfulfilled withdrawals, causing incorrect cross-chain/off-chain accounting (e.g., double conversion, wrong valuation when USDC/USDT diverge, or incorrect comparisons against USDC-denominated fields). This can lead to incorrect liquidity and withdrawal-handling decisions such as over/underestimating queued withdrawals, incorrect bridge sizing, and potential DoS or disruption of fulfillment workflows.

Recommendation

Consider making `VaultState` internally consistent. Convert `unfulfilledWithdrawalAmount` to the snapshot's `assetType` before returning from `getRemoteState()`, using the exact same converter/oracle, scaling, and rounding rules used for totalAssets/idleAssets.

Alleviation

[SuperEarn, 02/09/2026]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement. Moreover, the team responded as following:

We are already aware of this issue, and it is an intentional design choice.

At the moment, only USDT can be bridged, and the `unfulfilledAmount` represents the amount of USDT that actually needs to be sent to Kaia. If a conversion to USDC were to occur at this stage, we would no longer be able to determine how much USDT would ultimately be transferred through the bridge.

For other assets, USDC is used as the denominator because the majority of our strategies are primarily operated on a USDC basis.

For these reasons, while we acknowledge that the current structure is not perfectly clean, we are already aware of this trade-off and have decided not to modify it.

SUA-03 | Missing Check Actual Received Debt Payment

Category	Severity	Location	Status
Volatile Code	Minor	src/superearn/core/CooldownVault.sol (Dec-19): 742	Resolved

Description

The `retrieveDebt` retrieves debt payment from a strategy using the following statement:

```
742    uint256 debtPayment = strategy.repayPredepositDebt(predepositId);
```

Since the contract allows the governance role to add arbitrary strategies through the `addStrategy` function, the behavior and trustworthiness of each strategy cannot be guaranteed. Without verifying that the contract actually receives the `debtPayment` amount returned by `repayPredepositDebt`, there is a risk that a malicious or misconfigured strategy could misreport the amount, leading to incorrect accounting or loss of funds.

So it is important to validate that the received asset balance matches the reported `debtPayment` value to prevent discrepancies.

Recommendation

It's recommended that the implementation verifies the actual amount of assets received by the vault following the call to `strategy.repayPredepositDebt`.

Alleviation

[SuperEarn, 01/05/2026]: The team heeded the advice and resolved the issue by checking the actual received amount of assets in commit bba4483e6a45d64f19133f56415389671daf56d3.

SUA-04 | Missing Check For Feed Configuration

Category	Severity	Location	Status
Volatile Code	Minor	src/superearn/v2/core/vaults/OriginVault.sol (Dec-19): 309, 341, 352~353	Resolved

Description

The `totalAssets()` is the vault's TVL view function. Its call chain includes `convertAssetToUsdt()`, which requires `feedProxy` and `priceConverter` to be set to non-zero addresses. If these parameters are not configured, `totalAssets()` may revert, potentially blocking normal vault operations.

Recommendation

Ensure all required configuration is set before enabling operations.

```
275  if (feedProxy == address(0)) revert InvalidFeedProxy();
276  if (address(priceConverter) == address(0)) revert InvalidPriceConverter();
277  _unpause();
```

Alleviation

[SuperEarn, 01/04/2026]: The team heeded the advice and resolved the issue by adding two configuration checks in commit e19fbcd609a8c1508d845c247d426a3b54672ae7.

SUA-08 The `retrieveDebt` Health Check Is Evaluated In The Wrong Context And With Misleading Debt Parameters

Category	Severity	Location	Status
Logical Issue	Minor	src/superearn/core/CooldownVault.sol (Dec-19): 783; src/superearn/core/HealthCheck.sol (Dec-19): 116~122; src/superearn/core/USDOKycedCA.sol (Dec-19): 490	Acknowledged

Description

The `retrieveDebt` function invokes `healthCheck.check()` from the vault context. As a result, `msg.sender` within `HealthCheck` / `CommonHealthCheck` resolves to the vault address rather than the strategy address. This behavior may be inconsistent with the intended design of `HealthCheck.sol`.

As indicated by the notice in `HealthCheck.sol` `@notice Health check implementation for strategy harvest validation`, the health check appears to be designed to validate individual strategy behavior. This is further supported by the call to `IStrategyHealthCheck(customCheck).check(profit, loss, debtPayment, debtOutstanding, totalDebt, caller);`, which suggests the health check is expected to be performed on a per-strategy basis, rather than at the vault level.

CooldownVault.sol:

```
796         if (!healthCheck.check(profit, loss, _debtPayment, 0, _totalDebt)) {
797             revert HealthCheckFailed(profit, loss, _debtPayment, 0, _totalDebt)
;
798         }
```

HealthCheck.sol:

```
104     function check(
105         uint256 profit,
106         uint256 loss,
107         uint256 debtPayment,
108         uint256 debtOutstanding,
109         uint256 totalDebt
110     )
111         external
112         view
113         override(CommonHealthCheck, IHealthCheck)
114         returns (bool)
115     {
116         address caller = msg.sender;
117         address customCheck = checks[caller];
```

Additionally, `_checkHealth` passes (`debtOutstanding` = 0, `totalDebt` = `vault.totalDebt`). For partial repayments, `debtOutstanding` is non-zero (the `shortfall`), and using the vault-wide `totalDebt` as the denominator can dilute the loss threshold versus the intended per-strategy/per-predeposit basis. As a result, a large underpayment can pass even when health checks are believed to be enforcing tighter strategy-specific limits, leading to unexpected user losses/shortfall acceptance.

CooldownVault.sol:

```
796         if (!healthCheck.check(profit, loss, _debtPayment, 0, _totalDebt)) {
797             revert HealthCheckFailed(profit, loss, _debtPayment, 0, _totalDebt)
;
798         }
```

Similar finding exists in the `USDOKycedCA.sol` contract when claiming:

```
503         if (!healthCheck.check(profit, loss, _debtPayment, 0, _totalDebt)) {
504             revert HealthCheckFailed(profit, loss, _debtPayment, 0, _totalDebt)
;
505         }
```

Recommendation

Consider the below fixes:

(a) redesign the health-check interface to accept/verify the strategy address explicitly,

(b) have the strategy perform the health-check call (so `msg.sender` is the strategy) and return the result to the vault, and pass correct `debtOutstanding` / `totalDebt` values for the intended policy.

Alleviation

[SuperEarn, 01/16/2026]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement, instead, the team created a `GeneralHealthCheck` contract(out of current audit scope) to evaluate the status of non-strategy contracts. It works by calling external view functions on the health check target.

SUA-09 | Unnecessary `receive()` Function In `OriginVault`

Category	Severity	Location	Status
Inconsistency	Minor	src/superearn/v2/core/vaults/OriginVault.sol (Dec-19): 870	Resolved

Description

The `OriginVault` contract includes a `receive()` function, intended to "accept native tokens for CCIP fees". While this function exists, the cross-chain messaging and bridging architecture of the SuperEarn protocol dictates that CCIP fees are handled by the `adapter`, where the `RunespearSender._sendCCIPMessage()` handles the actual CCIP fee payment. The fee payment is deducted from `address(this)` in the context of `RunespearSender`, which is inherited by `CrosschainAdapter`. Therefore, the `adapter` (not the vault) is responsible for funding CCIP operations.

Consequently, sending native tokens to the `OriginVault`'s `receive()` function for the purpose of CCIP fees is meaningless within the protocol's design. Any native tokens sent to this function would not be utilized for CCIP messages and can only be recovered by the governance.

Recommendation

Consider removing the `receive()` and `emergencyRecoverETH()` functions from the `OriginVault` contract.

Alleviation

[SuperEarn, 01/12/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/superearn-io/superearn-core/commit/84b31efc3102eb5607af0b2cb5e9893101386441.

SUA-12 | Outbound Queue DoS Via Duplicate Nonces

Category	Severity	Location	Status
Logical Issue	Minor	src/superearn/v2/core/crosschain/BridgeQueue.sol (Dec-19): 213~236	Acknowledged

Description

The function `recordSentOperation` blindly appends whatever nonce the adapter supplies to `outboundAwaitingPeerReceipt`; there is no check that the nonce is not already queued.

The function `_removeOutboundAwaitingPeerReceipt` later relies on the `outboundAwaitingPeerReceiptIndex` mapping to know which slot to delete, so it can only delete the most recently written occurrence of a nonce. Any earlier duplicates lose their index entry (the mapping is overwritten and then deleted) and can never be removed.

A party that can call the `recordOutbound` flow (which is explicitly meant for “external adapter provided” nonces) can repeatedly re-use the same nonce before the peer confirms it. Each confirmation clears only the newest entry but leaves one stale copy behind with `peerReceiptRecorded = true`. Over time the array length grows without bound even though no operations remain pending.

Every management function that linearly scans this array such as `clearExpiredOutboundAwaitingPeerReceipt`, `manualClearOutboundByAmount`, `getExpiredCount`, `getOutboundAwaitingPeerReceiptNonces`, etc. incurs the cost of all of those unreachable entries.

With enough duplicates, these calls will exceed the block gas limit, permanently preventing governance from clearing genuinely stuck operations and causing `totalInTransit` to stay erroneously high (a live-lock/DoS of bridge recovery).

Recommendation

Before pushing, use `_getOutboundAwaitingPeerReceiptIndex` to ensure the nonce is not already awaiting receipt and revert if it is.

Alleviation

[SuperEarn, 01/23/2026]: `recordOutbound` in `BridgeAccountant.sol` is legacy function, this logic can be removed. (including `recordSentOperation` in `BridgeQueue.sol`).

This logic is never called now, and logic of `recordSentOperation` is replaced to `initiateBridge`.

SUA-13 Potential Fund Extraction Via `predeposit()` And `instantRedeem()`

Category	Severity	Location	Status
Logical Issue	Minor	src/superearn/core/CooldownVault.sol (Dec-19): 682~683, 832~833	Acknowledged

Description

A `strategy` can mint shares via `predeposit()` and immediately redeem them through `instantRedeem()` to withdraw idle vault assets, effectively reducing the contract's underlying token balance without providing upfront capital.

Specifically, calling `predeposit(assets)` mints an equivalent amount of shares to the strategy and increases `totalDebt` and `strategyDebtOutstanding` by `assets`. At this stage, no assets are transferred into the vault, and `_managedAssets` remains unchanged.

The strategy can then call `instantRedeem()` using the newly minted shares. This burns the shares and transfers an equivalent amount of assets from the vault to the strategy, provided sufficient idle assets are available. During this process, `_decreaseManagedAssets` is invoked, materializing the liquidity outflow.

If the strategy is compromised or misconfigured, this mechanism can be abused to drain vault liquidity. If the strategy does not subsequently repay the debt by calling `retrieveDebt`, user redemptions may face insufficient liquidity, resulting in underpayment recorded in `totalClaimLoss`. Governance can later call `recoverClaimLoss` to mint shares to itself to cover the accounting deficit, while affected users receive no direct compensation.

Recommendation

We raise this finding to remind the team to carefully manage strategy behavior and ensure that all operations remain consistent with the intended contract logic. If the team prefers to refactor the contract to further constrain this behavior, we recommend introducing an expiry mechanism for the `predeposit()` operation or restricting other strategy operations while outstanding debt remains unpaid.

Alleviation

[SuperEarn, 01/12/2026]:

We have carefully considered the proposed approaches, and our conclusion is that it is preferable not to modify the code at this time. Since all strategies are internally developed and operated by the team, we assume that intentionally malicious behavior will not occur.

In practice, predeposit is primarily triggered in conjunction with withdrawals. Given Yearn's design, assets withdrawn from a strategy are typically re-deposited into the same strategy. If an external strategy enforces a cooldown period (e.g., 7 days),

the predeposit would remain unfinalized for the duration of that cooldown. As a result, additional deposits could be blocked during this period, and if withdrawals occur frequently in small amounts, deposits could become effectively impossible.

In line with the intent of the comment, we acknowledge and understand the associated risk. We will ensure that this consideration is taken into account when expanding or introducing new strategies, and that it is carefully addressed during future strategy development.

SUA-14 | Missing Check On `receiver`

Category	Severity	Location	Status
Volatile Code	Minor	src/superearn/core/CooldownVault.sol (Dec-19): 409~410	Resolved

Description

The `redeem()` function allows an authorized user to request redemption of shares and specify a `receiver` for the underlying assets. However, it does not validate that `receiver` is non-zero.

If a redemption request is created with `receiver = address(0)`, the shares are burned while the assets become effectively locked. In this case, the `claim` path guarded by `ONLY_RECEIVER` for higher `maxLossBps` can never be satisfied. Additionally, during claiming the contract attempts to execute `safeTransfer(address(0), assetsOut)`, which typically reverts for standard ERC20 tokens, rendering the request permanently unclaimable. For tokens that allow transfers to the zero address, the underlying assets may instead be irreversibly burned.

Recommendation

It is recommended to check `receiver != address(0)` when creating redemption requests.

Alleviation

[SuperEarn, 01/12/2026]: The team heeded the advice and resolved the issue by checking `receiver != address(0)` in commit 1d7ae5d88b10cbeea0b319a2d22dca3b13cc709e.

SUA-15 Gas Exhaustion Risk Due To Unbounded Loops In `reconcile()`

Category	Severity	Location	Status
Logical Issue	Minor	src/superearn/v2/core/crosschain/BridgeQueue.sol (Dec-19): 836~841	Resolved

Description

The function `reconcile()` performs multiple full scans over `peerReceivedNonces`, `peerOutboundNonces`, and the entire `inboundAwaitingPeerRelease` list, and also calls `_deduplicateNonces()` which is O(n^2).

Because the reconciliation is executed on every inbound message, growth in pending nonce arrays can push reconcile over the block gas limit, preventing message handling/reconciliation and potentially halting cross-chain operations until state is manually reduced.

Recommendation

Consider removing quadratic deduplication, enforcing bounded snapshot sizes, and/or adding batched/paginated reconciliation.

Alleviation

[SuperEarn, 01/12/2026]: Since this part is handled by the internal keeper logic, the nonce array is not expected to grow excessively unless there is a significant operational mistake. We are already aware of this behavior, and it should be manageable with proper operational care.

SUA-16 | Missing Revoke Allowance When The External Call Fails

Category	Severity	Location	Status
Volatile Code	Minor	src/superearn/core/StrategyMorphoV1Vault.sol (Dec-19): 92~95	Resolved

Description

The `requestDeposit` function attempts to deposit assets into the external vault `metaMorpho`. If this external call fails, the catch block only sets the `success` flag to `false` but does not reset or clear the token allowance that was previously granted to `metaMorpho`.

As a result, the external contract retains the allowance to transfer tokens, which could create unintended security or operational risks, such as unauthorized spending if `metaMorpho`'s contract logic or permissions change.

Recommendation

It's recommended the token allowance granted to the external vault (`metaMorpho`) is reset or cleared in the catch block if the deposit attempt fails.

Alleviation

[SuperEarn, 01/13/2026]: The team heeded the advice and resolved the issue by resetting allowance to prevent unauthorized spending if deposit fails in commit c282eba29eea7ed582e578ede714b298637025fc.

SUA-28 `SYNC_BRIDGED` Can Be Spoofed Via `sendMessage` Without Funds/Accounting

Category	Severity	Location	Status
Logical Issue	Minor	src/superearn/v2/core/crosschain/CrosschainAdapter.sol (Dec-19): 602~625	Resolved

Description

The function `sendMessage()` restricts management and keeper to emit only `SYNC_NOOP`, and lets `vault/agent/governance` emit arbitrary predicates, including `SYNC_BRIDGED`, without running bridge accounting or moving funds.

```
611      if (
612          !(
613              msg.sender == vault || msg.sender == address(agent) || isGovernance
(msg.sender)
614                  || ((isManagement(msg.sender) || isKeeper(msg.sender)) &&
 predicate == RunespearProtocol.SYNC_NOOP)
615                  )
616              ) {
617                  revert UnauthorizedCaller();
618              }
```

Inbound function `_handle()` treats any `SYNC_BRIDGED` identically, if insufficient balance, enqueues a pending “awaiting assets” entry in the accountant.

A governance(misconfigured or compromised), can fabricate bridge notifications with no real transfer. The peer adapter will consider that funds are in flight, polluting the awaiting-asset queue and potentially requiring manual intervention or forced processing to clear.

Recommendation

It is recommended to restrict the emission of **SYNC_BRIDGED** so that it is triggered only when funds are actually transferred out.

Alleviation

[SuperEarn, 01/19/2026]: The team heeded the advice and resolved the issue by restricting `SYNC_BRIDGED` message in commit a39d122454a5d3a8ed5fcdb3a866c6b7de88dc78.

SUA-29 External Self-Call Pattern Risks Gas Starvation Of Outer CCIP Receive Frame

Category	Severity	Location	Status
Logical Issue	Minor	src/superearn/v2/messaging/runespear/RunespearReceiver.sol (Dec-19): 150~153	Resolved

Description

The function `_ccipReceive` uses `try this.processMessage()` (an external self-call) and then performs additional work in the outer frame.

```
149         // DEFENSIVE PATTERN: Try to process message, save if it fails
150 @>      try this.processMessage(message, sourceChainId) {
151             // Success - mark as processed
152 @>          processedMessages[message.messageId] = true;
153         } catch (bytes memory reason) {
154             // Failure - save for retry
155             _saveFailedMessage(message, sourceChainId, reason);
156             emit MessageFailed(message.messageId, sourceChainId, sender);
157             // DO NOT revert - message is saved and can be retried
158         }
```

Because the subcall is forwarded essentially all remaining gas by default, `processMessage/_handle` can consume nearly the entire CCIP gas stipend and still return successfully, leaving insufficient gas for the outer-frame SSTORE.

If the outer frame runs out of gas, the whole transaction reverts, undoing all subcall state changes and defeating the “defensive non-reverting” design.

Recommendation

Reserve gas for the outer frame, or move the `processedMessages` marking into `processMessage` itself.

Alleviation

[SuperEarn, 01/20/2026]: The team heeded the advice and resolved the issue by moving the marking logic into `processMessage()` in commit 321129cf6351f8806c72d416383b02e0ab9f4042. Moreover, a similar pattern exists in the failed-message handling path, that flow is executed through off-chain logic with a manually controlled gas limit. As a result, we do not expect the same issue to arise there.

SUA-30 | Callback Management Advertised But Not Enforced

Category	Severity	Location	Status
Inconsistency	Minor	src/superearn/v2/messaging/runespear/RunespearLib.sol (Dec-19): 165~181; src/superearn/v2/messaging/runespear/RunespearSender.sol (Dec-19): 198	Acknowledged

Description

The function `_sendMessage()` builds a fresh message, hardcoding `callbackHintHash = VOID_CALLBACK` and `remainingRounds = DEFAULT_MAX_ROUNDS`, discarding any hint from the original request.

Callback metadata is written and exposed by `isVoidCallback()/getCallbackHintHash()`, but the receive path (RunespearReceiver/CrosschainAdapter/agents) never reads or enforces it, so the “callback management” promise is unused metadata.

```
165    /**
166       * @notice Check if a message expects no callback
167       * @param message The message to check
168       * @return True if the message expects no callback
169       */
170      function isVoidCallback(Message memory message) internal pure returns (bool
) {
171          return message.header.callbackHintHash == VOID_CALLBACK;
172      }
173
174      /**
175       * @notice Get the callback hint hash from a message
176       * @param message The message to check
177       * @return The callback hint hash indicating expected async behavior
178       */
179      function getCallbackHintHash(Message memory message) internal pure returns
(bytes32) {
180          return message.header.callbackHintHash;
181      }
```

Recommendation

Either enforce/use `callbackHintHash` on the receive path or remove the hint field/functions to avoid false assurances and simplify the protocol.

Alleviation

[SuperEarn, 01/20/2026]: The team removed this feature in commit 10f58de41e6ccdcfe70e1e810298793113891ca6.

SUA-31 | Third-Party Dependency Usage

Category	Severity	Location	Status
Design Issue	Minor	src/superearn/core/HealthCheck.sol (Jan-21): 4~5; src/superearn/core/lib/TimelockExecutionLib.sol (Jan-21): 133~166; src/superearn/core/strategy/BaseCooldownStrategy.sol (Jan-21): 11~12, 11~12; src/superearn/core/strategy/StrategyMorphoV1Vault.sol (Jan-21): 11~12; src/superearn/v2/core/crosschain/CrosschainAdapter.sol (Jan-21): 21~22, 794~814; src/superearn/v2/core/vaults/RemoteVault.sol (Jan-21): 16~17, 51~52; src/superearn/v2/periphery/AssetPriceConverter.sol (Jan-21): 11~12, 11~22; src/superearn/v2/periphery/OraklAssetPriceConverter.sol (Jan-21): 4~5	Acknowledged

Description

The contracts are serving as the underlying entities to interact with one or more third party protocols, such as `Chainlink CCIP`, `@metamorpho`, `@yearn-vaults`. The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of third parties can possibly create severe impacts, such as causing incorrect price calculations, denial of service or fund loss.

`@openzeppelin/contracts-upgradeable/` library is used by the following contracts as third-party dependencies:

- `SuperEarnAccessControl`
- `SuperEarnMessageAgent`
- `CrosschainAdapter`
- `BridgeAccountant`
- `RemoteVault`
- `OriginVault`
- `OriginVaultBase`
- `RunespearTransceiver`
- `RunespearSender`
- `AssetPriceConverter`
- `OraklAssetPriceConverter`
- `CooldownVault`
- `USDOKycedCA`

`@openzeppelin/contracts/` library is used by the following contracts as third-party dependencies:

- `SuperEarnMessageAgent`
- `CrosschainAdapter`

- `StrategyOriginVault`
- `BaseCooldownStrategy`
- `CooldownVault`
- `OriginVault`
- `StrategyMorphoV1Vault`
- `StrategyUSDOExpressV2`
- `USDOKycedCA`
- `IStrategy`
- `IUSDOKycedCA`
- `AvalonUSDTVaultAPI`
- `ICooldownVault`
- `IVault`
- `RunespearSender`

`@chainlink/contracts-ccip/` is used by the following contracts as third-party dependencies:

- `CrosschainAdapter`
- `RunespearReceiver`
- `RunespearSender`

`@chainlink/contracts/` (price feeds) is used by the following contracts as third-party dependencies:

- `AssetPriceConverter`

`@metamorpho/` is used by the following contracts as third-party dependencies:

- `StrategyMorphoV1Vault`

`@yearn-vaults/` is used by the following contracts as third-party dependencies:

- `BaseCooldownStrategy`
- `HealthCheck`

`@uniswap/` is used by the following contracts as third-party dependencies:

- `AssetPriceConverter`

`@superearn/external/orakl/` is used by the following contracts as third-party dependencies:

- `OraklAssetPriceConverter`

Recommendation

The auditors understood that the business logic requires interaction with third parties. It is recommended for the team to constantly monitor the statuses of third parties to mitigate the side effects when unexpected activities are observed.

Alleviation

[SuperEarn, 02/02/2026]: The team acknowledged the issue and to mitigate it, they have a monitoring system in place.

Due to the nature of the protocol, funds are transferred through multiple transactions, and at each step, the results and execution status are actively monitored off-chain.

SUA-42 Inconsistent Reporting Of Redeemed Shares In `RemoteVault.withdrawFromYearn()`

Category	Severity	Location	Status
Inconsistency	Minor	src/superearn/v2/core/vaults/RemoteVault.sol (Dec-19): 496~503	Resolved

Description

When a keeper invokes `RemoteVault.withdrawFromYearn()`, the function delegates the withdrawal to `_withdrawFromYearn`, which processes the request by calling `superEarnRouter.redeem()`. After the call, the function sets `ySharesRedeemed = yShares` without verifying the actual number of yVault shares redeemed, then emits the `YearnRedemptionInitiated` event and returns this value.

However, Yearn's underlying vault logic may redeem fewer shares than requested if the vault lacks sufficient idle liquidity. In practice, the router tracks the real number of redeemed shares by comparing balances pre- and post-withdrawal, refunding any unredeemed shares to the caller.

This inconsistency means that the `YearnRedemptionInitiated` event and the returned value from `_withdrawFromYearn` may inaccurately reflect the number of shares actually redeemed by Yearn. This behavior contradicts interface documentation stating that the value represents the actual yVault shares redeemed, which could mislead off-chain indexers and users relying on this event for accounting and reconciliation.

Recommendation

It's recommended that the contract measures the actual number of yVault shares redeemed by calculating the difference between the pre- and post-withdrawal balances after calling `superEarnRouter.redeem`. Update both the `ySharesRedeemed` variable and the `YearnRedemptionInitiated` event to reflect this accurate value, ensuring the function return and emitted event consistently report the real number of shares redeemed.

Alleviation

[SuperEarn, 01/26/2026]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/superearn-io/superearn-core/commit/09bff8c86b024474f140580914f395e6f6b75345.

SUA-43 The `emergencyWithdraw` Function Does Not Exclude The `usdc` And The `usdo`

Category	Severity	Location	Status
Volatile Code	Minor	src/superearn/core/USDOKycedCA.sol (Dec-19): 955~961	Acknowledged

Description

The `emergencyWithdraw()` function bypasses the `totalRedeemedUsdcAmt` reservation mechanism and allows governance to fully withdraw `USDC` or `USDO` while the contract is paused. As a result, funds reserved for pending redemptions can be drained, potentially leaving the protocol underfunded and making redemptions impossible once the contract is unpaused.

Additionally, although the function's comment explicitly states that governance must not withdraw tokens reserved for pending redeems, this restriction is not enforced in the actual implementation. The absence of a corresponding check in code means that the protection exists only as a comment and can be violated either accidentally or maliciously.

Recommendation

Consider restricting the `emergencyWithdraw` function, if those tokens(`usdc/usdo`) must be recoverable, enforce amount <= balance - `totalRedeemedUsdcAmt()` (and the analogous `totalRedeemedUsdoAmt`) before transferring.

Alleviation

[SuperEarn, 01/26/2026]: This contract is unlikely to hold USDO in practice. For USDC, withdrawals excluding the reserved amount are already possible via the recover function.

Although the comment on this function warns against withdrawing funds reserved for pending redeems, the actual scenarios in which this function is intended to be used are limited. In practice, it would be called when USDC is still held by the contract but claims cannot be processed due to an issue in the strategy logic. In such cases, the purpose of this function is to forcibly withdraw funds, including amounts pending claim, to prevent funds from becoming permanently stuck.

For these reasons, we believe the current implementation is appropriate and should remain as is.

SUA-44 | Incorrect Event Emission

Category	Severity	Location	Status
Logical Issue	Minor	src/superearn/v2/core/vaults/OriginVault.sol (Dec-19): 470~473, 529~532	Resolved

Description

The `withdrawFromRemote()` function checks whether the `agent` address is non-zero before calling `agent.sendMessage()`. However, regardless of whether the `agent` is the zero address, the `RemoteWithdrawalRequested` event is always emitted.

This behavior can mislead off-chain services or users that rely on this event, as it may be interpreted as confirmation that a remote withdrawal has been initiated. In cases where the `agent` address is the zero address, no message is sent and no withdrawal is actually requested, yet the event still signals otherwise.

Note that the `processRedemptionQueue()` function also emits the `RemoteWithdrawalRequested` event when the `agent` address is zero.

Recommendation

It's recommended that the team updates the `withdrawFromRemote` function so that the `RemoteWithdrawalRequested` event is only emitted when a remote withdrawal is actually initiated (i.e., when the agent is not the zero address and `sendMessage` is called). This will help ensure that off-chain services and users can reliably interpret emitted events as indicators of real actions taken by the contract.

Alleviation

[SuperEarn, 01/26/2026]: The team heeded the advice and resolved the issue by emitting the event inside the `if` branch in commit 99a70994e2c0ee410eb7c32b4d75918dea78baf5.

SUA-45 Emergency Token Recovery Allows Draining Of Vault Share Balances

Category	Severity	Location	Status
Volatile Code	Minor	src/superearn/v2/core/vaults/OriginVault.sol (Dec-19): 849	Resolved

Description

The `emergencyRecoverToken()` function only prevents recovery of the underlying `asset`, but does not block recovery of the vault's own ERC-20 share token. When users call `requestRedeem()`, their share tokens are transferred to `address(this)` and later burned during redemption. If governance calls `emergencyRecoverToken()` with `token == address(this)`, all shares held by the vault can be withdrawn.

This breaks the redemption flow, as subsequent redeem operations attempt to burn share tokens that the contract no longer holds, causing redemptions to revert and effectively locking users out.

Recommendation

Consider adding a guard such as `if (token == address(this)) revert CannotRecoverVaultShares();` to prevent emergency recovery from withdrawing the vault's own share tokens.

Alleviation

[SuperEarn, 01/26/2026]: The team heeded the advice and resolved the issue by adding the code snippet `if (token == address(this)) revert CannotRecoverVaultShares();` in commit 1ad5b7e71684926d0e432e476da7fa066809b2d9.

SUA-50 | Allowlist Is Only Enforced At Submission Not At Execution

Category	Severity	Location	Status
Logical Issue	Minor	src/superearn/core/lib/TimelockExecutionLib.sol (Feb-2): 97~98, 133~134	Resolved

Description

The function `submitExecution()` validates `allowedTargets`, but the `acceptExecution()` does not re-check the allowlist. If a target is removed from the `allowedTargets` after a proposal is queued (e.g., target compromise), the pending execution can still call that target.

Governance may still execute calls to a target that was removed in an emergency.

Recommendation

Re-validate the allowlist in `acceptExecution()`, or clear pending executions when the `setAllowedTarget()` removes a target.

Alleviation

[SuperEarn, 02/09/2026]: The team heeded the advice and resolved the issue in commit 4840f99ed90fc2bed37678c67cd87f399da989bb.

SUA-53 Changing `bridgeDepositAddress` Can Strand Previously Deposited Funds

Category	Severity	Location	Status
Design Issue	Minor	src/superearn/v2/core/crosschain/CrosschainAdapter.sol (Feb-2): 231~232, 374~375	Acknowledged

Description

The `setBridgeDepositAddress()` simply replaces `bridgeDepositAddress` without checking the previous deposit address's state of assets. After the change, `sendAssets()` continues sending to the new address, the concern is what if the former address still hold tokens still belong to the protocol. Because governance is allowed to update the address at any time, assets may be stranded whenever a new deposit address is configured while the old one still holds bridged funds.

Recommendation

Disallow updating `bridgeDepositAddress` unless the old address has a zero balance (or is explicitly withdrawn/migrated first).

Alleviation

[SuperEarn, 02/09/2026]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement. Moreover, the team responded that, in the current implementation, the deposit address is issued and managed by the bridge service we are using, so we cannot directly withdraw funds from it or migrate them ourselves.

Although we have an established communication channel with the team and trust them, there is still a potential risk of a bottleneck if the bridge experiences an issue or if, for any reason, funds cannot be released from that address. For this reason, we intend to keep the current code as is.

SUA-56 | `defaultGasLimit` Remains Uninitialized

Category	Severity	Location	Status
Logical Issue	Minor	src/superearn/v2/messaging/runespear/RunespearSender.sol (Feb-2): 55~56; src/superearn/v2/messaging/runespear/RunespearTransceiver.sol (Feb-2): 19~20	Resolved

Description

The `RunespearTransceiver` contract combines bidirectional Runespear communication logic into a single contract. Its initializer, `__RunespearTransceiver_init()`, invokes `__RunespearSender_init(router, feeToken)` to initialize the `RunespearSender` component. However, neither initializer explicitly sets `RunespearSender.defaultGasLimit`.

Although `defaultGasLimit` is defined with an inline state initializer:

```
uint256 public defaultGasLimit = 500_000;
```

this assignment is not applied when the contract is deployed behind a proxy. Since the contract suite follows an upgradeable pattern using `Initializable` and `_disableInitializers()`, Solidity inline state initializers are not executed in proxy storage. As a result, `defaultGasLimit` remains uninitialized and defaults to `0`.

Recommendation

It is recommended to explicitly initialize `defaultGasLimit` during initialization.

Alleviation

[SuperEarn, 02/09/2026]: The team heeded the advice and resolved the issue by moving `defaultGasLimit = 500_000;` into `__RunespearSender_init()` in commit e375802adbcc9c111437de51538e80dd2e6b9ef9.

SUA-06 Function Calls User-Provided Addresses With No Access Control Modifier

Category	Severity	Location	Status
Access Control	Informational	src/superearn/periphery/SuperEarnRouter.sol (Dec-19): 168	Resolved

Description

Calling a user provided address is dangerous, especially in a public function with no access control restriction. An attacker could deploy a malicious contract and use the vulnerable function to trigger a call to the malicious contract, potentially stealing user funds or causing other serious damages.

Recommendation

We recommend several different types of mitigations, depending on the context:

1. Remove the vulnerable function, or restrict what addresses can be called from it.
2. Include access control mechanisms, whether it be through making the function `internal` or restricting which contracts can call this function.

Alleviation

[SuperEarn, 01/05/2026]: The team heeded the advice and resolved the issue by adding whitelist for `yVault` in commit bba4483e6a45d64f19133f56415389671daf56d3.

SUA-10 | Missing ERC4626 `Deposit` Events

Category	Severity	Location	Status
Code Optimization	Informational	src/superearn/core/CooldownVault.sol (Dec-19): 228~229, 331~332, 352~353	Acknowledged

Description

The `deposit()`, `mint()`, and `depositFor()` functions mint `cooldownVault` shares to the receiver in exchange for an equivalent amount of underlying tokens, overriding the behavior defined in `IERC4626Upgradeable`.

While `depositFor()` in `ERC20WrapperUpgradeable` emits the ERC20 `Transfer` event, it does **not** emit the `Deposit` event required by the `IERC4626` standard:

```
event Deposit(address indexed sender, address indexed owner, uint256 assets, uint256 shares);
```

As a result, the absence of this event makes deposits difficult to track on-chain.

Recommendation

It is recommended to emit the `IERC4626.Deposit` event in both `deposit` and `mint` flows.

Alleviation

[SuperEarn, 01/12/2026]: Although CooldownVault implements the ERC4626 interface, it does not inherit from an actual ERC4626 implementation. The contract was initially developed following the ERC4626 model; however, as the internal logic of CooldownVault evolved, it became incompatible with strict ERC4626 semantics. As a result, the implementation was refactored to use ERC20Wrapper instead, while retaining the ERC4626 interface for backward compatibility with existing integrations and prior development work.

Accordingly, we acknowledge this finding and do not plan to make changes at this time.

SUA-18 | Unused Variables

Category	Severity	Location	Status
Coding Style	Informational	src/superearn/core/StrategyMorphoV1Vault.sol (Dec-19): 55; src/superearn/core/StrategyOriginVault.sol (Dec-19): 134; src/superearn/core/StrategyUSDOExpressV2.sol (Dec-19): 42	Resolved

Description

The `oneERC4626` in `StrategyMorphoV1Vault.sol` is a **private** immutable variable that is never used after assignment.

The `oneAsset` in `StrategyOrigin.sol` is a **private** immutable variable that is never used after assignment.

The `oneUsdo` in `StrategyUSDOExpressV2.sol` is a **private** immutable variable that is never used after assignment.

Recommendation

Consider removing unused variables.

Alleviation

[SuperEarn, 01/13/2026]: The team heeded the advice and resolved the issue by commenting out the unused immutable private variables in commit e505d5531ddee33b16cd1b38850a98765b70df0c.

SUA-20 Potential Denial Of Service In Strategy Liquidation Via `liquidateAllPositions()`

Category	Severity	Location	Status
Denial of Service	Informational	src/superearn/core/StrategyMorphoV1Vault.sol (Jan-13): 296~297	Acknowledged

Description

The `liquidateAllPositions()` function is designed to liquidate the strategy's entire position by calling `premintCooldownVault()` with an amount (`estimatedTotalAssets() + oneUsdt`) intended to cover all assets.

Inside `premintCooldownVault()` , the function calculates the number of shares to redeem based on this requested amount. Due to the input exceeding the total assets, the `Math.min` operation sets the redeem `shares` to the strategy's entire balance of `externalShareToken` (`MetaMorpho` shares). The function then calls `requestRedeem(shares)` .

The vulnerability is that `requestRedeem()` attempts to call `metaMorpho.redeem(shares, ...)` for the full balance without checking the available liquidity. If the `MetaMorpho` vault has insufficient balance, the `redeem()` call reverts. The `requestRedeem()` function catches this revert in a `try-catch` block and returns `false` .

As a result, `premintCooldownVault()` returns early with zeros, and `liquidateAllPositions()` returns `0` . This logic causes a Denial of Service (DoS) where the strategy fails to free *any* assets if it cannot free *all* of them.

Recommendation

We raise this issue to highlight the potential DoS risk and to confirm with the team whether this behavior has been fully considered and is an intentional design choice. If this behavior is not intended, we recommend refactoring the logic to account for the available liquidity in `MetaMorpho` prior to calling `metaMorpho.redeem(shares, ...)` . Specifically, the strategy should allow partial redemptions so that `liquidateAllPositions()` can return a non-zero amount even when the underlying liquidity in `MetaMorpho` is insufficient for a full redemption.

Alleviation

[SuperEarn, 01/22/2026]: We confirm the reported behavior. We also observed that, even when maxRedeem indicates all shares are redeemable, the transaction can still revert if the Morpho position lacks sufficient underlying balance at execution time.

Given this, it may be better not to modify liquidateAllPositions. Instead, we should handle cases where full redemption is not immediately available through operational controls, such as reducing the strategy's debt ratio or adjusting the withdrawal queue.

SUA-22 Inconsistent Handling Of `premintCooldownVault()` Return Values Across Cooldown Strategies

Category	Severity	Location	Status
Logical Issue, Design Issue	Informational	src/superearn/core/StrategyUSDOExpressV2.sol (Dec-19): 280~281, 295~296	Resolved

Description

Both the `StrategyOriginVault` and `StrategyUSDOExpressV2` strategies inherit from `BaseCooldownStrategy` and configure a non-zero `cooldownPeriod`.

However, the two strategies handle the return values of `premintCooldownVault()` differently.

`StrategyOriginVault` overrides `premintCooldownVault()` and explicitly tracks unbacked redemption shares using `totalUnbackedRedemptionShares`. These unbacked shares are later incorporated into asset accounting via the `estimatedTotalAssets()` function.

In particular, if a redemption is not backed by a successful predeposit, the corresponding redemption shares are added to `totalUnbackedRedemptionShares`, which directly affects the strategy's asset estimation logic.

By contrast, `StrategyUSDOExpressV2` simply calls the parent contract's `premintCooldownVault()` implementation without inspecting or acting on its return values. As a result, it does not track unbacked redemption shares, and its `estimatedTotalAssets()` implementation differs accordingly.

Given that both strategies appear to follow similar business logic and share the same cooldown-related constraints, we would like to request clarification from the team on why they intentionally implement different return-value handling and validation logic when calling `premintCooldownVault()` in the parent contract.

Recommendation

Consider checking the design and the implementation and aligning the them if necessary.

Alleviation

[SuperEarn, 01/29/2026]: The team heeded the advice and resolved the issue by removing the unreachable branches in commit f5b4ed3ac694d60eba44e09aff50502675ff4eb3.

SUA-32 | Incorrect Comment Implies Optional `SYNC_BRIDGED`

Category	Severity	Location	Status
Coding Style	Informational	src/superearn/v2/core/crosschain/CrosschainAdapter.sol (Dec-19): 352	Resolved

Description

In the function `sendAssets()`, the comment says “Optionally send a SYNC_BRIDGED message,” but the implementation always sends `SYNC_BRIDGED`, no conditional branch.

Misleading documentation can cause operators/auditors to assume `SYNC_BRIDGED` is optional.

Recommendation

Update the comment to reflect current behavior.

Alleviation

[SuperEarn, 01/19/2026]: The team heeded the advice and resolved the issue by removing the misleading comment in commit 23ab336b02aa8bf268a73a7a8d9717cb2128f73b.

SUA-33 | Redundant Statements

Category	Severity	Location	Status
Volatile Code	Informational	src/superearn/v2/core/crosschain/CrosschainAdapter.sol (Dec-19): 111, 234~238	Resolved

Description

The linked statements do not affect the functionality of the codebase and appear to be either remnants of test code or older functionality.

```
    mapping(address => bool) public authorizedBridges;

    function setAuthorizedBridge(address bridge, bool authorized) external override
onlyGovernance {
        if (bridge == address(0)) revert InvalidAddress();
        authorizedBridges[bridge] = authorized;
        emit AuthorizedBridgeSet(bridge, authorized);
    }
```

Recommendation

We recommend removing this redundant code to improve clarity and ensure the implementation is better suited for production environments.

Alleviation

[SuperEarn, 01/19/2026]: The team heeded the advice and resolved the issue in commit 7f2cf197d1202e0f2bf16fbca6b73da253c5ca74.

SUA-34 | Unused Message Processing Hooks

Category	Severity	Location	Status
Volatile Code	Informational	src/superearn/v2/messaging/runespear/RunespearReceiver.sol (Dec-19): 267, 273; src/superearn/v2/messaging/runespear/RunespearSender.sol (Dec-19): 333, 339	Resolved

Description

`_beforeProcessMessage` , `_afterProcessMessage` , `_beforeSendMessage` , and `_afterSendMessage` are virtual hook functions intended to allow child contracts to inject custom logic around message handling.

However, these functions are left unimplemented and unused, and could either be removed or properly implemented to improve clarity.

Recommendation

Consider removing the unused hook functions if they are not expected to be used, or add documentation or example implementations to indicate their intended use.

Alleviation

[SuperEarn, 01/19/2026]: The team heeded the advice and resolved the issue by removing the unused hook functions in commit 88f012be76b11391565c3dfe2e1588a3d4354b20.

SUA-35 Missing Implementation Of Declared Function `deactivateChain()`

Category	Severity	Location	Status
Inconsistency	Informational	src/superearn/v2/messaging/runespear/RunespearReceiver.sol (Dec-19): 122~126; src/superearn/v2/messaging/runespear/RunespearSender.sol (Dec-19): 119~123	Resolved

Description

The `RunespearSender` contract is responsible for sending cross-chain messages, while the `RunespearReceiver` contract processes incoming messages. They both contain incomplete function declarations:

- RunespearSender.sol (lines 119-122): "@notice Deactivate a chain configuration"
- RunespearReceiver.sol (lines 123-125): "@notice Deactivate a source chain"

These unimplemented function placeholders prevent chain configurations from being deactivated, which could be problematic in emergency situations or when chains need to be disabled.

Recommendation

Implement the missing deactivation functions in both contracts.

Alleviation

[SuperEarn, 01/24/2026]: The team heeded the advice and resolved the issue by removing the code comments in commit 10f58de41e6ccdcfe70e1e810298793113891ca6.

SUA-36 | Dead Code

Category	Severity	Location	Status
Coding Style	Informational	src/superearn/core/StrategyOriginVault.sol (Dec-19): 625~635	Resolved

Description

The `adjustPosition` function updates the `toInvestShares` based on the `toInvestShares`, when the `toInvestShares` is either zero or `type(uint256).max`, the `toInvestShares` will be updated. The `toInvestShares` is calculated from the `getSupplyCap()` that always assigns `type(uint256).max` to the `toInvestShares`:

```
    //StrategyOriginVault.sol
    function adjustPosition(uint256 _debtOutstanding) internal virtual override
nonReentrant {
            ..
            (, uint256 _availableAssets) = getSupplyCap();  //<@
            if (_availableAssets == 0) {
                toInvestShares = 0;
            } else if (_availableAssets != type(uint256).max) {
                try cooldownVault.previewDeposit(_availableAssets) returns (uint256
maxSharesFromCap) {
                    toInvestShares = Math.min(toInvestShares, maxSharesFromCap);
                }
                ..
            }
            ..
    }


    function getSupplyCap() public view virtual override returns (uint256
supplyAssetsCap, uint256 availableAssets) {
        availableAssets = originVault.maxDeposit(address(this));
        supplyAssetsCap = availableAssets;
    }
```

```
    //OriginVaultBase.sol
    function maxDeposit(address /* receiver */ ) public view virtual returns
(uint256) {
        return type(uint256).max;
    }
```

As a result, the `if` branch and the `else-if` branch in the adjustPosition will never be executed and becomes dead code.

Recommendation

It is recommended to review both the implementation and the overall design, and to remove any dead code if it is no longer needed or serves no functional purpose.

Alleviation

[SuperEarn, 01/28/2026]: The team heeded the advice and resolved the issue by removing the dead code in commit 8b7ab715bcc68145176d8b2b1d396d2236c12626.

SUA-37 | The `USDOKycedCA` Calls An Only-Whitelist Function

Category	Severity	Location	Status
Design Issue	Informational	src/superearn/core/USDOKycedCA.sol (Dec-19): 178~179	Acknowledged

Description

The `_mint` function relies on calling the only-whitelist function `instantMint` of the external `usdoExpress` contract(out of current audit scope), `instantMint()` enforces that both the caller (`from`) and the recipient (`to`) are included in an internal whitelist (`_kycList`): **USDOExpressV2.sol**

```
    function instantMint(address underlying, address to, uint256 amt) external
whenNotPausedMint {
        address from = _msgSender();
        if (!_kycList[from] || !_kycList[to]) revert USDOExpressNotInKycList(from,
to);

        (uint256 usdoAmtCurr, uint256 fee) = _instantMintInternal(underlying, from,
to, to, amt);
        emit InstantMint(underlying, from, to, amt, usdoAmtCurr, fee);
    }

    function grantKycInBulk(address[] calldata _addresses) external
onlyRole(WHITELIST_ROLE) {
        for (uint256 i = 0; i < _addresses.length; i++) {
            _kycList[_addresses[i]] = true;
        }
        emit USDOKycGranted(_addresses);
    }

    function revokeKycInBulk(address[] calldata _addresses) external
onlyRole(WHITELIST_ROLE) {
        for (uint256 i = 0; i < _addresses.length; i++) {
            _kycList[_addresses[i]] = false;
        }
        emit USDOKycRevoked(_addresses);
    }
```

Recommendation

Control over this whitelist is centralized, as the `grantKycInBulk` function allows addresses to be added by an entity with the `WHITELIST_ROLE` in the `usdoExpress` contract. This introduces a dependency on the third-party's management of KYC status, meaning that changes in their whitelist can directly affect the ability of `USDOKycedCA` to mint as intended. If the

whitelist status is removed, restricted, or managed in a way that is inconsistent with project requirements, it could disrupt the contract's operations.

Alleviation

[SuperEarn, 01/26/2026]: We are aware of this point, and since we have an established working relationship with the OpenEden team and an active communication channel, we do not expect this to become a significant issue.

SUA-39 Discussion On Incomplete Position Adjustment In `StrategyMorphoV1Vault`

Category	Severity	Location	Status
Logical Issue	Informational	src/superearn/core/strategy/StrategyMorphoV1Vault.sol (Jan-21): 320, 323~324	Acknowledged

Description

The `adjustPosition()` function computes the surplus `want` balance and deposits it into `MetaMorpho`. Its execution flow is as follows:

1. The function determines the amount of `want` shares eligible for investment by taking the current `want` balance, subtracting the vault's `_debtOutstanding`, and capping the resulting value by both `maxInstantRedeem` and the external supply cap (converted to shares).
2. The investable `want` balance is then converted to USDT (`toInvestAssets`), and `requestDeposit(toInvestAssets)` is called to deposit into `MetaMorpho`, which returns the amount of assets actually deposited (`filledAssets`).
3. The unfilled amount is calculated as the difference between `toInvestAssets` and `filledAssets`.
4. Only this unfilled portion is deposited into the `cooldownVault` for conversion back into `want` tokens, while any pre-existing USDT balance is ignored.

As a result, the function processes only the USDT that failed to be deposited into `MetaMorpho` and does not account for any USDT already held by the strategy. This behavior may lead to reduced capital utilization.

Recommendation

We recommend confirming with the client whether this design is intentional and whether the implications of ignoring pre-existing USDT balances have been fully considered.

Alleviation

[SuperEarn, 02/05/2026]: We understand this point. From our perspective, a pre-existing USDT balance at the time adjustPosition() is executed should not occur unless it is caused by an external donation. If you see any other plausible flow that could result in such a balance, we would appreciate it if you could point it out. Otherwise, the current implementation behaves as intended.

SUA-46 | Lack Of Reasonable Upper Boundaries

Category	Severity	Location	Status
Logical Issue	Informational	src/superearn/core/CooldownVault.sol (Dec-19): 1014~1020; src/superearn/v2/core/vaults/RemoteVault.sol (Dec-19): 293~296	Acknowledged

Description

The `setMaxSlippage()` function only enforces that `_maxSlippageBps` is less than or equal to `BASIS_POINTS`. However, `BASIS_POINTS` (10,000) is not a reasonable upper bound for `maxSlippageBps`, as it allows the value to be set to 100%. Permitting a maximum slippage of 100% disables slippage protection and is not considered best practice.

Recommendation

Introduce an explicit upper bound for `maxSlippageBps` in `setMaxSlippage()` to ensure the value remains within acceptable limits, rather than allowing it to be set as high as 100%.

Alleviation

[SuperEarn, 01/26/2026]: Slippage here effectively serves as a health check. Even if the slippage is set to a high value, it does not introduce an attack vector. In that case, increasing the slippage would effectively have the same effect as temporarily disabling the health check.

SUA-52 | Broken Partial Claim Logic Prevents Redeeming With Slippage

Category	Severity	Location	Status
Logical Issue	Informational	src/superearn/core/CooldownVault.sol (Feb-2): 500	Resolved

Description

The `_initiateRedemption()` function automatically attempts to claim assets when `cooldownPeriod` is 0 by calling `_claim` with a `minAssetsOut` calculated from `maxLossThresholdBps`. However, the `_claim` function contains a logical flaw in how it validates asset availability against `reservedForPriorRequests`.

```
 if (cooldownPeriod == 0) {
            uint256 minAssetsOut = MathUpgradeable.mulDiv(assets, BASIS_POINTS -
maxLossThresholdBps, BASIS_POINTS);
            // ignore failure; can claim later after liquidity becomes available
@>          _claim(requestId, minAssetsOut);
        }
```

Inside `_claim`, `minAssetsOut` is only used at the very end as a lower-bound check for the “minimum acceptable payout” specified by the user. Before that, `assetsOut` is set as:

```
@>   assetsOut = MathUpgradeable.min(request.assets, _managedAssets);
@>   if (assetsOut < minAssetsOut) {
         return ("EXCESSIVE_LOSS", assetsOut);
     }
```

It is then added to `reservedForPriorRequests` and compared against total liquidity `_managedAssets`:

```
   ...
     uint256 reservedForPriorRequests =
                _accRedeemRequestedAmount > accClaimedAmount ?
_accRedeemRequestedAmount - accClaimedAmount : 0;

     if (assetsOut + reservedForPriorRequests > _managedAssets) {
        return ("INSUFFICIENT_ASSETS", 0);
     }
```

This check effectively requires that the full amount of the current request can be paid immediately without consuming any assets reserved for earlier requests.

As a result, once there is any request ahead in the queue, subsequent requests cannot rely on slippage protection to receive a partial payout.

Scenario

1. Assume a `CooldownVault` with `cooldownPeriod` set to 0 and `maxLossThresholdBps` set to 500 (5% slippage allowed).
2. User A makes a redemption request for 100 assets. The vault currently lacks liquidity, so User A's request remains pending. `accRedeemRequestedAmount` for User A is 100.
3. The vault receives 195 assets (e.g., from strategy repayment). `_managedAssets` = 195. Note that 100 is reserved for User A, leaving 95 available.
4. User B calls `redeem` for 100 assets, and is willing to accept 95 assets (5% loss).
5. `_initiateRedemption` calls `_claim` for User B.
6. Inside `_claim`:
    - `reservedForPriorRequests` is 100 (for User A).
    - `assetsOut` is calculated as `min(100, 195) = 100`.
    - The check `100 (assetsOut) + 100 (reserved) > 195 (_managedAssets)` evaluates to `200 > 195`, which is true.
7. The claim fails with `INSUFFICIENT_ASSETS`, even though 95 unreserved assets are available and User B was willing to accept 95. User B is locked in the vault unnecessarily.

Recommendation

If the intended design requires each request to be either fully satisfied or not executed at all, the current behavior is consistent with that goal. However, if the intention is to allow requests to receive partial payouts based on available unreserved liquidity and slippage constraints, the current logic does not support this behavior. We would like to confirm the intended design with the team to ensure the implementation consistently aligns with the business logic.

Alleviation

[SuperEarn, 02/11/2026]: In general, we provide users with a permissionless claim mechanism that does not allow loss under normal conditions.

If a user chooses to increase maxLossBps and execute a direct claim, this effectively creates an exit path where the user voluntarily accepts potential loss based on their own judgment. This behavior is intentional by design.

Regarding L597, checking only request.assets <= _managedAssets may appear inconsistent with the failure message. However, the error condition (assetsOut < minAssetsOut) implicitly assumes that reservedForPriorRequests > 0.

For these conditions to be satisfied simultaneously, the available liquidity must already be reduced due to reserved amounts. In other words, although request.assets is less than or equal to managedAssets, once the reserved portion is deducted from managedAssets, the effective available liquidity becomes insufficient - which justifies the "INSUFFICIENT_ASSETS" error.

SUA-57 | Permanent Fund Lock Due To Missing Controller Validation

Category	Severity	Location	Status
Logical Issue	Informational	src/superearn/v2/core/vaults/OriginVault.sol (Feb-2): 601~602, 781~782	Resolved

Description

The redemption flow enforces strict access control in `redeem()` via `if (controller != msg.sender && !isOperator[controller][msg.sender])`. However, `requestRedeem()` allows an arbitrary `controller` address without validating that it is non-zero or capable of initiating calls.

As a result, a user can invoke `requestRedeem(shares, address(0), owner)`, which successfully creates a redemption request and transfers or locks the shares. Vault operators may later process this request through `processRedemptionQueue` or `batchFulfillRedemptions`, crediting `redemptionRequests[item.controller].lockedAssets` and `fulfilledAssets`, and incrementing `totalFulfilledRedemptionAssets`, without verifying that the specified controller can ever call `redeem()`.

Because transactions cannot originate from `address(0)` and no operator can be approved on behalf of `address(0)`, any attempt to claim the redemption via `redeem(requestId, receiver, address(0))` will revert with `InvalidCaller`. This results in both the shares and the fulfilled assets being permanently locked.

Recommendation

Consider validating the controller in the `requestRedeem()`:

- Reject `controller == address(0)`.

Alleviation

[SuperEarn, 02/09/2026]: The team heeded the advice and resolved the issue by rejecting the zero address of the controller in commit 9b456ded1a7b4df02be3f5d6bc12b051de12708c.

SUA-58 | Incorrect Minimum-Length Check

Category	Severity	Location	Status
Logical Issue	Informational	src/superearn/v2/messaging/runespear/RunespearLib.sol (Feb-2): 65~66	Resolved

Description

`decodeMessage()` checks `data.length < 68` as a “minimum size (header + predicate)”, but messages are produced via `abi.encode(header, predicate, args)`, whose minimum length (with empty `args`) is 160 bytes:

1. `protocolId`: 32 bytes (bytes32)
2. `version`: 32 bytes (uint8, ABI-padded)
3. `predicate`: 32 bytes (bytes4, ABI-padded)
4. `args`: 32 bytes offset + 32 bytes length

As a result, the malformed/short payloads pass the length check and then revert inside `abi.decode()` with a generic ABI decoding error/panic instead of the intended `InvalidMessageLength()` error, preventing cheap early rejection and making failure handling less predictable.

Recommendation

It is recommended to validate against the actual ABI encoding layout produced by `abi.encode(header, predicate, args)`.

Alleviation

[SuperEarn, 02/09/2026]: The team heeded the advice and resolved the issue by updating the length check from `data.length < 68` to `data.length < 160` in commit 7bf88f2c48f44182129abfb60637bc48e9722420.

SUA-59 The `redeem()` Does Not Clear Per-Request Amounts And The `claimableRedeemRequest()` Ignores Redeemed Flag

Category	Severity	Location	Status
Volatile Code	Informational	src/superearn/v2/core/vaults/OriginVault.sol (Feb-2): 645~646, 664~665, 781~782	Resolved

Description

After a successful `redeem()`, the contract only sets `item.redeemed=true` and decreases aggregate controller/global locked accounting, but it leaves `redemptionQueue[queueIndex].shares` and `.fulfilledAssets` unchanged.

Separately, `claimableRedeemRequest()` (and `pendingRedeemRequest()`) only checks `queueIndex` against `queueFulfilledIndex` and does not check `item.redeemed`.

As a result, already-claimed requests can continue to report as claimable in ERC-7540-style view calls, causing automated claimers/keepers/strategies that rely on `claimableRedeemRequest()` to repeatedly attempt `redeem()` and revert with `AlreadyRedeemed`, potentially breaking batch-claim flows and causing avoidable DoS for integrators.

Recommendation

To avoid stale or misleading state, consider updating `claimableRedeemRequest()` and `pendingRedeemRequest()` to return `0` once `item.redeemed == true`, and/or explicitly clearing `item.shares` and `item.fulfilledAssets` during `redeem()`.

Alleviation

[SuperEarn, 02/09/2026]: The team heeded the advice and resolved the issue by returning 0 once `item.redeemed == true` for the functions `claimableRedeemRequest()` and `pendingRedeemRequest()` in commit b4a97c94de074023161dbd223b12faa27dd6b289.

OPTIMIZATIONS | SUPEREARN - AUDIT

ID	Title	Category	Severity	Status
SUA-19	Inefficient Message Decoding In Extraction Helper Functions	Gas Optimization	Optimization	Resolved

SUA-19 | Inefficient Message Decoding In Extraction Helper Functions

Category	Severity	Location	Status
Gas Optimization	Optimization	src/superearn/v2/messaging/runespear/RunespearLib.sol (Dec-19): 188~191, 198~201	Resolved

Description

The `extractPredicate()` and `extractArgs()` helper functions both call `decodeMessage()` to fully decode the entire message structure, even though they only need to extract a single field. This results in unnecessary gas consumption and computational overhead.

When both functions are called sequentially, and the message is decoded twice, doubling the gas cost unnecessarily.

Recommendation

It is recommended to optimize these functions or streamline the function invocation process to eliminate unnecessary gas costs.

Alleviation

[SuperEarn, 01/19/2026]: The team heeded the advice and resolved the issue by removing these functions in commit 321129cf6351f8806c72d416383b02e0ab9f4042.

FORMAL VERIFICATION | SUPEREARN - AUDIT

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied formal verification to prove that important functions in the smart contracts adhere to their expected behaviors.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of contracts derived from AccessControl v4.4

We verified properties of the public interface of contracts that provide an AccessControl-v4.4 compatible API. This involves:

- The `hasRole` function, which returns `true` if an account has been granted a specific `role`.
- The `getRoleAdmin` function, which returns the admin role that controls a specific `role`.
- The `grantRole` and `revokeRole` functions, which are used for granting a `role` to an account and revoking a `role` from an `account`, respectively.
- The `renounceRole` function, which allows the calling account to revoke a `role` from itself.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
accesscontrol-hasrole-succeed-always	`hasRole` Function Always Succeeds
accesscontrol-default-admin-role	AccessControl Default Admin Role Invariance
accesscontrol-getroleadmin-change-state	`getRoleAdmin` Function Does Not Change State
accesscontrol-hasrole-change-state	`hasRole` Function Does Not Change State
accesscontrol-getroleadmin-succeed-always	`getRoleAdmin` Function Always Succeeds
accesscontrol-renouncerole-revert-not-sender	`renounceRole` Reverts When Caller Is Not the Confirmation Address
accesscontrol-grantrole-correct-role-granting	`grantRole` Correctly Grants Role
accesscontrol-renouncerole-succeed-role-renouncing	`renounceRole` Successfully Renounces Role
accesscontrol-revokerole-correct-role-revoking	`revokeRole` Correctly Revokes Role

Verification Results

For the following contracts, formal verification established that each of the properties that were in scope of this audit (see scope) are valid:

Detailed Results For Contract AssetPriceConverter (src/superearn/v2/periphery/AssetPriceConverter.sol) In Commit 70fa63512fd794b17a55d13bc840243740ee063c

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	True	
accesscontrol-hasrole-change-state	True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-change-state	True	
accesscontrol-getroleadmin-succeed-always	True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	True	
accesscontrol-renouncerole-succeed-role-renouncing	True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	True	

Detailed Results For Contract BridgeAccountant (src/superearn/v2/core/crosschain/BridgeAccountant.sol) In Commit 70fa63512fd794b17a55d13bc840243740ee063c

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	True	
accesscontrol-renouncerole-succeed-role-renouncing	True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	True	
accesscontrol-getroleadmin-change-state	True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	True	
accesscontrol-hasrole-change-state	True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	True	

APPENDIX | SUPEREARN - AUDIT

Audit Scope

superearn-io/superearn-core
src/superearn/core/CooldownVault.sol
src/superearn/core/HealthCheck.sol
src/superearn/core/USDOKycedCA.sol
src/superearn/v2/core/crosschain/BridgeQueue.sol
src/superearn/v2/core/vaults/OriginVault.sol
src/superearn/v2/core/vaults/RemoteVault.sol
src/superearn/v2/messaging/runespear/RunespearLib.sol
src/superearn/v2/messaging/runespear/RunespearReceiver.sol
src/superearn/v2/messaging/runespear/RunespearSender.sol
src/superearn/v2/messaging/runespear/RunespearTransceiver.sol
src/superearn/core/lib/TimelockExecutionLib.sol
src/superearn/core/strategy/BaseCooldownStrategy.sol
src/superearn/core/strategy/StrategyMorphoV1Vault.sol
src/superearn/core/strategy/StrategyOriginVault.sol
src/superearn/core/CooldownVault.sol
src/superearn/core/HealthCheck.sol
src/superearn/core/minter/USDOKycedCA.sol
src/superearn/v2/base/SuperEarnAccessControl.sol
src/superearn/v2/core/crosschain/BridgeAccountant.sol

superearn-io/superearn-core
src/superearn/v2/core/crosschain/CrosschainAdapter.sol
src/superearn/v2/core/crosschain/SuperEarnMessageAgent.sol
src/superearn/v2/core/vaults/OriginVault.sol
src/superearn/v2/core/vaults/OriginVaultBase.sol
src/superearn/v2/core/vaults/RemoteVault.sol
src/superearn/v2/periphery/AssetPriceConverter.sol
src/superearn/v2/periphery/OraklAssetPriceConverter.sol
src/superearn/core/minter/USDOKycedCA.sol
src/superearn/v2/core/crosschain/CrosschainAdapter.sol
src/superearn/v2/libraries/VaultStateHelper.sol
src/superearn/core/BaseCooldownStrategy.sol
src/superearn/core/StrategyMorphoV1Vault.sol
src/superearn/core/StrategyOriginVault.sol
src/superearn/core/StrategyUSDOExpressV2.sol
src/superearn/v2/core/crosschain/BridgeAccountant.sol
src/superearn/v2/core/crosschain/CrosschainAdapter.sol
src/superearn/v2/periphery/AssetPriceConverter.sol
src/superearn/core/strategy/StrategyUSDOExpressV2.sol
src/superearn/core/lib/TimelockExecutionLib.sol
src/superearn/core/CooldownVault.sol
src/superearn/v2/core/vaults/OriginVault.sol
src/superearn/v2/messaging/runespear/RunespearTransceiver.sol

superearn-io/superearn-core
src/superearn/v2/messaging/runespear/RunespearSender.sol
src/superearn/v2/messaging/runespear/RunespearLib.sol
src/superearn/core/lib/TimelockExecutionLib.sol
src/superearn/interface/IBaseFeeOracle.sol
src/superearn/interface/ICooldownVault.sol
src/superearn/interface/IERC7540.sol
src/superearn/interface/IERC7575.sol
src/superearn/interface/IHealthCheck.sol
src/superearn/interface/IRegistry.sol
src/superearn/interface/IStrategy.sol
src/superearn/interface/IStrategyCooldownAware.sol
src/superearn/interface/ISuperEarnRouter.sol
src/superearn/interface/IVault.sol
src/superearn/interface/IERC6900ExecutionHookLightModule.sol
src/superearn/interface/IUSDOKycedCA.sol
src/superearn/v2/base/SuperEarnAccessControl.sol
src/superearn/v2/core/crosschain/SuperEarnMessageAgent.sol
src/superearn/v2/core/vaults/OriginVaultBase.sol
src/superearn/v2/interfaces/IBridgeAccountant.sol
src/superearn/v2/interfaces/ICrosschainAdapter.sol
src/superearn/v2/interfaces/ICrosschainVault.sol
src/superearn/v2/interfaces/IOriginVault.sol

superearn-io/superearn-core
src/superearn/v2/interfaces/IRemoteVault.sol
src/superearn/v2/interfaces/IRunespearAgent.sol
src/superearn/v2/interfaces/IRunespearReceiver.sol
src/superearn/v2/interfaces/ISwapQuoter.sol
src/superearn/v2/libraries/VaultStateHelper.sol
src/superearn/v2/messaging/runespear/RunespearProtocol.sol
src/superearn/v2/messaging/SuperEarnV2Protocol.sol
src/superearn/v2/periphery/OraklAssetPriceConverter.sol
src/superearn/api/USDOExpressV2API.sol
src/superearn/api/USDOExpressV2API.sol
src/superearn/interface/IBaseFeeOracle.sol
src/superearn/interface/ICooldownVault.sol
src/superearn/interface/IERC6900ExecutionHookLightModule.sol
src/superearn/interface/IERC7540.sol
src/superearn/interface/IERC7575.sol
src/superearn/interface/IHealthCheck.sol
src/superearn/interface/IRegistry.sol
src/superearn/interface/IStrategy.sol
src/superearn/interface/IStrategyCooldownAware.sol
src/superearn/interface/ISuperEarnRouter.sol
src/superearn/interface/IUSDOKycedCA.sol
src/superearn/interface/IVault.sol

superearn-io/superearn-core
src/superearn/v2/core/crosschain/BridgeQueue.sol
src/superearn/v2/interfaces/ICrosschainAdapter.sol
src/superearn/v2/interfaces/ICrosschainVault.sol
src/superearn/v2/interfaces/IOriginVault.sol
src/superearn/v2/interfaces/IBridgeAccountant.sol
src/superearn/v2/interfaces/IRemoteVault.sol
src/superearn/v2/interfaces/IRunespearAgent.sol
src/superearn/v2/interfaces/IRunespearReceiver.sol
src/superearn/v2/interfaces/ISwapQuoter.sol
src/superearn/v2/libraries/VaultStateHelper.sol
src/superearn/v2/messaging/runespear/RunespearLib.sol
src/superearn/v2/messaging/runespear/RunespearProtocol.sol
src/superearn/v2/messaging/runespear/RunespearReceiver.sol
src/superearn/v2/messaging/runespear/RunespearSender.sol
src/superearn/v2/messaging/runespear/RunespearTransceiver.sol
src/superearn/v2/messaging/SuperEarnV2Protocol.sol
src/superearn/api/USDOExpressV2API.sol
src/superearn/core/lib/TimelockExecutionLib.sol
src/superearn/core/strategy/BaseCooldownStrategy.sol
src/superearn/core/strategy/StrategyMorphoV1Vault.sol
src/superearn/core/strategy/StrategyOriginVault.sol
src/superearn/core/strategy/StrategyUSDOExpressV2.sol

superearn-io/superearn-core
src/superearn/core/CooldownVault.sol
src/superearn/core/HealthCheck.sol
src/superearn/interface/IBaseFeeOracle.sol
src/superearn/interface/ICooldownVault.sol
src/superearn/interface/IERC6900ExecutionHookLightModule.sol
src/superearn/interface/IERC7540.sol
src/superearn/interface/IERC7575.sol
src/superearn/interface/IHealthCheck.sol
src/superearn/interface/IRegistry.sol
src/superearn/interface/IStrategy.sol
src/superearn/interface/IStrategyCooldownAware.sol
src/superearn/interface/ISuperEarnRouter.sol
src/superearn/interface/IUSDOKycedCA.sol
src/superearn/interface/IVault.sol
src/superearn/v2/base/SuperEarnAccessControl.sol
src/superearn/v2/core/crosschain/BridgeAccountant.sol
src/superearn/v2/core/crosschain/BridgeQueue.sol
src/superearn/v2/core/crosschain/CrosschainAdapter.sol
src/superearn/v2/core/crosschain/SuperEarnMessageAgent.sol
src/superearn/v2/core/vaults/OriginVault.sol
src/superearn/v2/core/vaults/OriginVaultBase.sol
src/superearn/v2/core/vaults/RemoteVault.sol

superearn-io/superearn-core
src/superearn/v2/interfaces/IBridgeAccountant.sol
src/superearn/v2/interfaces/ICrosschainAdapter.sol
src/superearn/v2/interfaces/ICrosschainVault.sol
src/superearn/v2/interfaces/IOriginVault.sol
src/superearn/v2/interfaces/IRemoteVault.sol
src/superearn/v2/interfaces/IRunespearAgent.sol
src/superearn/v2/interfaces/IRunespearReceiver.sol
src/superearn/v2/interfaces/ISwapQuoter.sol
src/superearn/v2/libraries/VaultStateHelper.sol
src/superearn/v2/messaging/runespear/RunespearLib.sol
src/superearn/v2/messaging/runespear/RunespearProtocol.sol
src/superearn/v2/messaging/runespear/RunespearReceiver.sol
src/superearn/v2/messaging/runespear/RunespearSender.sol
src/superearn/v2/messaging/runespear/RunespearTransceiver.sol
src/superearn/v2/messaging/SuperEarnV2Protocol.sol
src/superearn/v2/periphery/AssetPriceConverter.sol
src/superearn/v2/periphery/OraklAssetPriceConverter.sol
src/superearn/core/minter/USDOKycedCA.sol
src/superearn/core/strategy/BaseCooldownStrategy.sol
src/superearn/core/strategy/StrategyMorphoV1Vault.sol
src/superearn/core/strategy/StrategyOriginVault.sol
src/superearn/core/strategy/StrategyUSDOExpressV2.sol

superearn-io/superearn-core
src/superearn/core/HealthCheck.sol
src/superearn/api/USDOExpressV2API.sol
src/superearn/v2/base/SuperEarnAccessControl.sol
src/superearn/v2/core/crosschain/BridgeAccountant.sol
src/superearn/v2/core/crosschain/BridgeQueue.sol
src/superearn/v2/core/crosschain/SuperEarnMessageAgent.sol
src/superearn/v2/core/vaults/OriginVaultBase.sol
src/superearn/v2/core/vaults/RemoteVault.sol
src/superearn/v2/interfaces/IBridgeAccountant.sol
src/superearn/v2/interfaces/ICrosschainAdapter.sol
src/superearn/v2/interfaces/ICrosschainVault.sol
src/superearn/v2/interfaces/IOriginVault.sol
src/superearn/v2/interfaces/IRemoteVault.sol
src/superearn/v2/interfaces/IRunespearAgent.sol
src/superearn/v2/interfaces/IRunespearReceiver.sol
src/superearn/v2/messaging/runespear/RunespearReceiver.sol
src/superearn/v2/messaging/runespear/RunespearProtocol.sol
src/superearn/v2/messaging/SuperEarnV2Protocol.sol
src/superearn/v2/periphery/AssetPriceConverter.sol
src/superearn/v2/periphery/OraklAssetPriceConverter.sol
src/superearn/interface/IBaseFeeOracle.sol
src/superearn/interface/ICooldownVault.sol

superearn-io/superearn-core
src/superearn/interface/IERC6900ExecutionHookLightModule.sol
src/superearn/interface/IERC7540.sol
src/superearn/interface/IERC7575.sol
src/superearn/interface/IHealthCheck.sol
src/superearn/interface/IRegistry.sol
src/superearn/interface/IStrategy.sol
src/superearn/interface/IStrategyCooldownAware.sol
src/superearn/interface/ISuperEarnRouter.sol
src/superearn/interface/IUSDOKycedCA.sol
src/superearn/interface/IVault.sol
src/superearn/api/USDOExpressV2API.sol
src/superearn/core/lib/TimelockExecutionLib.sol
src/superearn/core/minter/USDOKycedCA.sol
src/superearn/core/strategy/BaseCooldownStrategy.sol
src/superearn/core/strategy/StrategyMorphoV1Vault.sol
src/superearn/core/strategy/StrategyOriginVault.sol
src/superearn/core/strategy/StrategyUSDOExpressV2.sol
src/superearn/core/CooldownVault.sol
src/superearn/core/HealthCheck.sol
src/superearn/interface/IBaseFeeOracle.sol
src/superearn/interface/ICooldownVault.sol
src/superearn/interface/IERC6900ExecutionHookLightModule.sol

superearn-io/superearn-core
src/superearn/interface/IERC7540.sol
src/superearn/interface/IERC7575.sol
src/superearn/interface/IHealthCheck.sol
src/superearn/interface/IRegistry.sol
src/superearn/interface/IStrategy.sol
src/superearn/interface/IStrategyCooldownAware.sol
src/superearn/interface/ISuperEarnRouter.sol
src/superearn/interface/IUSDOKycedCA.sol
src/superearn/interface/IVault.sol
src/superearn/v2/base/SuperEarnAccessControl.sol
src/superearn/v2/core/crosschain/BridgeAccountant.sol
src/superearn/v2/core/crosschain/BridgeQueue.sol
src/superearn/v2/core/crosschain/CrosschainAdapter.sol
src/superearn/v2/core/crosschain/SuperEarnMessageAgent.sol
src/superearn/v2/core/vaults/OriginVault.sol
src/superearn/v2/core/vaults/OriginVaultBase.sol
src/superearn/v2/core/vaults/RemoteVault.sol
src/superearn/v2/interfaces/IBridgeAccountant.sol
src/superearn/v2/interfaces/ICrosschainAdapter.sol
src/superearn/v2/interfaces/ICrosschainVault.sol
src/superearn/v2/interfaces/IOriginVault.sol
src/superearn/v2/interfaces/IRemoteVault.sol

superearn-io/superearn-core
src/superearn/v2/interfaces/IRunespearAgent.sol
src/superearn/v2/interfaces/IRunespearReceiver.sol
src/superearn/v2/libraries/VaultStateHelper.sol
src/superearn/v2/messaging/runespear/RunespearLib.sol
src/superearn/v2/messaging/runespear/RunespearProtocol.sol
src/superearn/v2/messaging/runespear/RunespearReceiver.sol
src/superearn/v2/messaging/runespear/RunespearSender.sol
src/superearn/v2/messaging/runespear/RunespearTransceiver.sol
src/superearn/v2/messaging/SuperEarnV2Protocol.sol
src/superearn/v2/periphery/AssetPriceConverter.sol
src/superearn/v2/periphery/OraklAssetPriceConverter.sol

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Incorrect Calculation	Incorrect Calculation findings are about issues in numeric computation such as rounding errors, overflows, out-of-bounds and any computation that is not intended.
Denial of Service	Denial of Service findings indicate that an attacker may prevent the program from operating correctly or responding to legitimate requests.
Access Control	Access Control findings are about security vulnerabilities that make protected assets unsafe.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.

Categories	Description
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified. Each such contract was compiled into a mathematical model that reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The following assumptions and simplifications apply to our model:

- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held when it was invoked.
- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.

- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

Description of the Analyzed AccessControl-v4.4 Properties

Properties related to function `hasRole`

accesscontrol-hasrole-change-state

The `hasRole` function must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrol-hasrole-succeed-always

The `hasRole` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `DEFAULT_ADMIN_ROLE`

accesscontrol-default-admin-role

The default admin role must be invariant, ensuring consistent access control management.

Specification:

```
invariant DEFAULT_ADMIN_ROLE() == 0x00;
```

Properties related to function `getRoleAdmin`

accesscontrol-getroleadmin-change-state

The `getRoleAdmin` function must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrol-getroleadmin-succeed-always

The `getRoleAdmin` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `renounceRole`

accesscontrol-renouncerole-revert-not-sender

The `renounceRole` function must revert if the caller is not the same as `account`.

Specification:

```
reverts_when account != msg.sender;
```

accesscontrol-renouncerole-succeed-role-renouncing

After execution, `renounceRole` must ensure the caller no longer has the renounced role.

Specification:

```
ensures !hasRole(role, account);
```

Properties related to function `grantRole`

accesscontrol-grantrole-correct-role-granting

After execution, `grantRole` must ensure the specified account has the granted role.

Specification:

```
ensures hasRole(role, account);
```

Properties related to function `revokeRole`

accesscontrol-revokerole-correct-role-revoking

After execution, `revokeRole` must ensure the specified account no longer has the revoked role.

Specification:

```
ensures !hasRole(role, account);
```

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is the largest blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.