Security Review Report
NM-0314 LRT Square

NETHERMIND SECURITY

(Sep 23, 2024)

Contents

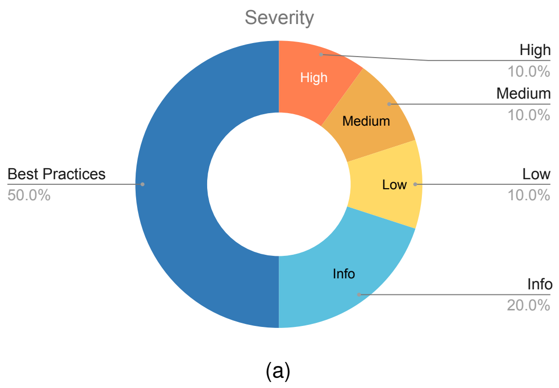
1 Executive Summary

This document presents the security review performed by Nethermind Security for LRT Square protocol smart contracts. LRT Square is a vault that holds the AVS tokens as underlying assets. Through LRT Square, LRT projects can pool AVS rewards into a single vault and issue vault share tokens to their stakers (or distributor contract). This system cuts transaction costs and streamlines the rewards process, which is great for those with smaller stakes who might find it easier to manage and trade their shares collectively. Larger stakeholders can redeem and possibly arbitrage, which will drive the market price of the LRT Square token.

The audited code comprises 943 lines of code written in the Solidity language. The audit included the Vault contract together with a PriceProvider and the implementation of one Swapper using the 1Inch protocol.

The audit was performed using (a) manual analysis of the codebase, (b) automated analysis tools, and (c) creation of test cases. **Along this document, we report** 10 points of attention, where one is classified as `High`, one is classified as `Medium`, one is classified as `Low`, and seven are classified as `Informational` or `Best Practice`. The issues are summarized in Fig. 1.

This document is organized as follows. Section 2 presents the files in the scope. Section 3 summarizes the issues. Section 4 presents the system overview. Section 5 discusses the risk rating methodology. Section 6 details the issues. Section 7 discusses the documentation provided by the client for this audit. Section 8 presents the compilation, tests, and automated tests. Section 9 concludes the document.



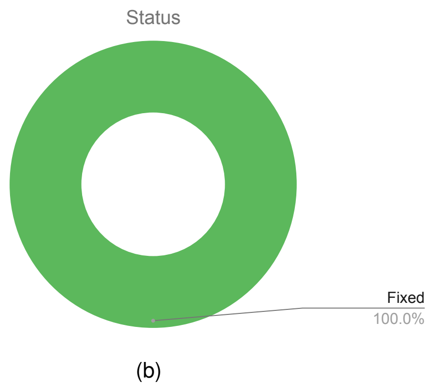


Fig. 1: Distribution of issues: Critical (0), **High** (1), **Medium** (1), **Low** (1), **Undetermined** (0), **Informational** (2), **Best Practices** (5).
Distribution of status: Fixed (10), **Acknowledged** (0), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	September 20, 2024
Response from Client	Regular responses during audit engagement
Final Report	September 23, 2024
Repository	lrt-square-sc
Commit (Audit - Section 2-I)	1045312ea148da02c6fb66213980bda243d5d804
Commit (Audit - Section 2-II)	5145e4e9585f4cc9e2facee2c2b7ab1236912405
Final Commit (Section 2-I)	fa7bb781ff1a05687fd17d66b5bec22f5d1c5b81
Final Commit (Section 2-II)	d356220ce021afaa97dafe33ed7507badc149ea2
Documentation	GitBook
Documentation Assessment	Medium
Test Suite Assessment	Medium

2 Audited Files

2.1 Part I

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/UUPSProxy.sol	8	1	12.5%	2	11
2	src/LrtSquare.sol	557	38	6.8%	113	708
3	src/Swapper1InchV6.sol	79	36	45.6%	13	128
4	src/PriceProvider.sol	115	2	1.7%	27	144
5	src/libraries/BucketLimiter.sol	71	84	118.3%	12	167
6	src/interfaces/IAggregatorV3.sol	28	1	3.6%	5	34
7	src/interfaces/IPriceProvider.sol	5	3	60.0%	2	10
8	src/interfaces/IOneInch.sol	37	35	94.6%	7	79
9	src/interfaces/ISwapper.sol	10	9	90.0%	1	20
	Total	**910**	**209**	**23.0%**	**182**	**1301**

2.2 Part II

The contract presented in the table below was audited considering the commit `5145e4e9585f4cc9e2facee2c2b7ab1236912405`.

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/Swapper1InchV6.sol	33	22	66%	10	65

3 Summary of Issues

	Finding	Severity	Update
1	The `getVaultTokenValuesInEth(...)` function only consider whitelisted tokens	High	Fixed
2	Weight limits are not checked after a `rebalance` operation	Medium	Fixed
3	Wrong computation to compute from shares to amounts and vice versa	Low	Fixed
4	Deposits may fail due to rounding errors	Info	Fixed
5	`getPriceInEth(...)` is called unnecessarily	Info	Fixed
6	Duplicated checking for registered tokens	Best Practices	Fixed
7	Tautology in the function `assetsForVaultShares` for `isTokenRegistered`	Best Practices	Fixed
8	Unused array in `_verifyPositionLimits(...)`	Best Practices	Fixed
9	Unused event	Best Practices	Fixed
10	`swap` function should implement reentrancy guard	Best Practices	Fixed

4 System Overview

AVSs can distribute the restaking rewards in any ERC20 token. However, most users will receive small, fractional amounts (more trouble than they're worth to claim or trade in the swap pools). It can lead to frustrating experiences and poor returns for stakers, while AVS may see a low token utilization and increased selling pressure.

LRT Square is a unified restaking rewards protocol as a vault that holds the rewards tokens as underlying assets. Through LRT Square, LRT projects can pool AVS rewards into a single vault and issue vault share tokens to their stakers (or distributor contract). This system cuts transaction costs and streamlines the rewards process, which is great for those with smaller stakes who might find it easier to manage and trade their shares collectively. Larger stakeholders can redeem and possibly arbitrage, which will drive the market price of the LRT Square token.

The vault includes the main three following flows:

- **deposit(...)** Registered depositors use this function to add tokens to the vault and mint new shares. Only whitelisted tokens can be deposited, and the value of one share must be kept through a *deposit* action. The *deposit* actions are capped to limit the amount of new shares that can be minted during a certain time-lapse. *Deposit* actions are also restricted by the weight limit associated with each token. These limits restrict the part of the total value that can be represented by that token.
- **redeem(...)** Any shareholder can use this function to redeem tokens stored in the vault. The caller will receive all the different tokens stored in the vault on a pro-rata basis.
- **rebalance(...)** This operation can only be called by the rebalancer role. The function is used to swap tokens held in the vault through the set *swapper* contract. Currently, there is only one swapper implementation that uses the *1Inch* protocol. The received token must be whitelisted. The swap is protected by a *maxSlippageForRebalancing* value guarding that the total vault value is not reduced by a relevant amount after a *rebalance* operation.

Besides the three described functions, the contract contains multiple other functions that can only be called by the governors. The following methods are used to configure the state of the vault:

- **mint(...)**
- **whitelistRebalacingOutputToken(...)**
- **setSwapper(...)**
- **registerToken(...)**
- **setRebalancer(...)**
- **updateWhitelist(...)**
- **updateTokenPositionWeightLimit(...)**
- **setDepositors(...)**
- **updatePriceProvider(...)**
- **setRefillRatePerSecond(...)**
- **setRateLimitCapacity(...)**
- **_authorizeUpgrade(...)**
- **setCommunityPauseDepositAmount(...)**

The contract also features two *pause* mechanisms. An account granted with the *PAUSER_ROLE* can *pause* and *unpause* the contract at any time. Besides this, any user can *pause* the contract through the function **communityPause(...)** with the requirement that the caller must send *depositForCommunityPause* amount of *ETH* to the contract, this amount will be after refunded to the caller if the reason of the pause was legit.

5 Risk Rating Methodology

The risk rating methodology used by Nethermind Security follows the principles established by the OWASP Foundation. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood measures how likely the finding is to be uncovered and exploited by an attacker. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploits the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage, such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage, such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage, such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding, other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Severity Risk		
Impact	**High**	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Info/Best Practices	Low	Medium
	Undetermined	Undetermined	Undetermined	Undetermined
		Low	**Medium**	**High**
		Likelihood		

To address issues that do not fit a High/Medium/Low severity, Nethermind Security also uses three more finding severities: **Informational**, **Best Practices**, and **Undetermined**.

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to pass to the client formally;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

c) **Undetermined** findings are used when we cannot predict the impact or likelihood of the issue.

6 Issues

6.1 [High] The `getVaultTokenValuesInEth(...)` function only consider whitelisted tokens

File(s): `src/LrtSquare.sol`

Description: The function `getVaultTokenValuesInEth(...)` computes the value in `eth` of the specified amount of vault tokens. The function computes the total value held in the vault nominated in `eth` and computes the value of each share.

```
function getVaultTokenValuesInEth(
    uint256 vaultTokenShares
) public view returns (uint256) {
    uint256 totalSupply = totalSupply();
    if (totalSupply == 0) {
        return 0;
    }

    (
        address[] memory assets,
        uint256[] memory assetAmounts
    // @audit - This function only return the whitelisted tokens
    ) = totalAssets();
    uint256 totalValue = getTokenValuesInEth(assets, assetAmounts);
    return (totalValue * vaultTokenShares) / totalSupply;
}
```

The problem arises because the `totalAssets(...)` function only returns the current whitelisted tokens, ignoring the rest of the registered tokens.

The wrong computation of the total value may cause multiple issues, including the following ones:

- The number of shares minted when depositing will be higher than what it should be because the vault's total value was not considered. When these shares are redeemed, they will return more value than expected;
- The weights of each token in the vault will not be correctly computed due to the wrong total value;
- Rebalance operations done by swapping non-whitelisted tokens to whitelisted tokens can effectively reduce the vault value because the swapped tokens were not taking in account before the trade;

Recommendation(s): Consider all the registered tokens when computing the value of the vault shares.

Status: Fixed.

Update from the client: Fixed in commit e85501addeea322a1160841ae34cb4e916adda53.

6.2 [Medium] Weight limits are not checked after a `rebalance` operation

File(s): `src/LrtSquare`

Description: Each registered token has a weight limit. This value restricts the vault value that this token can represent. After each `deposit` operation, it is checked that all the tokens are below their limits. However, `rebalance` operations change the distribution of tokens in the vault, and they do not check that the limits are enforced. A rebalance operation could increase the token amount in the vault even if the new amount is over the weight limit for that token.

If the vault is set to a distribution that does not respect the configured limits some deposit operations would be unavailable until the vault is rebalanced again.

Recommendation(s): Verify the position limits after a rebalance operation.

Status: Fixed.

Update from the client: Fixed in commit b2f4a06439eea6bce7e9383ea04102c7d2f63db2.

6.3 [Low] Wrong computation to compute from shares to amounts and vice versa

File(s): `src/LrtSquare`

Description: The `_convertToAssetAmount(...)` and `_convertToShares(...)` functions compute the amount of one asset corresponding to a number of shares and the number of shares corresponding to a set of assets.

```
function _convertToShares(
    uint256 valueInEth,
    Math.Rounding rounding
) public view virtual returns (uint256) {
    uint256 _totalSupply = totalSupply();
    return
        valueInEth.mulDiv(
            _totalSupply + 10 ** _decimalsOffset(),
            getVaultTokenValuesInEth(_totalSupply) + 1,
            rounding
        );
}

function _convertToAssetAmount(
    address assetToken,
    uint256 vaultShares,
    Math.Rounding rounding
) internal view virtual returns (uint256) {
    return
        vaultShares.mulDiv(
            IERC20(assetToken).balanceOf(address(this)) + 1,
            totalSupply() + 10 ** _decimalsOffset(),
            rounding
        );
}
```

The addition of `one` in both functions makes the result of the expression less accurate. In the case of the `_convertToShares(...)` function, the value is added to the denominator, causing the amount of shares being minted to be potentially lower. In the case of the `_convertToAssetAmount(...)` function, the value is added to the multiplier, causing the number of tokens returned to be potentially greater than it should be.

Recommendation(s): Remove the addition of `one` in both functions and update the code accordingly.

Status: Fixed.

Update from the client: Fixed in commit c3dc210f1310aef9443855acf4e4f3dcea9d3bf3.

6.4 [Info] Deposits may fail due to rounding errors

File(s): `src/LrtSquare.sol`

Description: The `deposit(...)` function checks that one share's value remains the same before and after deposits. However, due to rounding errors, it is possible that the value of shares increases in some edge cases. The call to `deposit(...)` would fail in the cases mentioned because the mentioned property will not hold.

Recommendation(s): Document these behaviors to make depositors aware of the possible failure. Optionally, consider changing this property to ensure that the value of shares does not decrease.

Status: Fixed.

Update from the client: Fixed in commit `8164a58df897016e6165532bef64c7e57bc1bbf0`.

6.5 [Info] `getPriceInEth(...)` is called unnecessarily

File(s): `src/LrtSquare.sol`

Description: The function `getTokenValuesInEth(...)` calls `getPriceInEth(...)` twice unnecesarily. This operation is potentially expensive, and removing the extra call could save a significant amount of gas.

```
function getTokenValuesInEth(...) public view returns (uint256) {
        ...
        uint256 price = IPriceProvider(priceProvider).getPriceInEth(
            _tokens[i]
        );
        if (price == 0) revert PriceProviderFailed();

        total_eth +=
            (_amounts[i] *
                // @audit - This call is unnecessary
                IPriceProvider(priceProvider).getPriceInEth(_tokens[i])) /
            10 ** _getDecimals(_tokens[i]);
    }
    return total_eth;
}
```

Recommendation(s): Remove the unnecessary call.

Status: Fixed.

Update from the client: Fixed in commit 1dc5c47f164f88ed29afa9b079a9325e161a3d50.

6.6 [Best Practice] Duplicated checking for registered tokens

File(s): `src/LrtSquare.sol`

Description: During the deposit process, the tokens to be deposited are checked if they are registered and whitelisted in the function `previewDeposit(...)` to compute `shareToMint` amount. Then, the tokens are transferred to the vault in `_deposit(...)`. The function `_deposit` checks again if the `_token[i]` is registered.

```
function _deposit(
    address[] memory _tokens,
    uint256[] memory amounts,
    uint256 shareToMint,
    address recipientForMintedShare
) internal {
    for (uint256 i = 0; i < _tokens.length; i++) {
        // @audit Duplicated check. It is also applied when
        //        shareToMint amount is calculated
        if (!isTokenRegistered(_tokens[i])) revert TokenNotRegistered();
    ...
}
```

Recommendation(s): Consider removing the duplicated checking `isTokenRegistered` in the internal function `_deposit`.

Status: Fixed.

Update from the client: Fixed in commit 4d99bd03cb9261c4000e61420b891c4730312213.

6.7 [Best Practice] Tautology in the function `assetsForVaultShares` for `isTokenRegistered`

File(s): `src/LrtSquare.sol`

Description: The function `assetsForVaultShares(...)` computes the asset amounts for all registered tokens. It checks if all assets in the array `tokens` are registered. However, `isTokenRegistered` always returns true for `tokens[i]`.

```
function assetsForVaultShares(
    uint256 vaultShare
) public view returns (address[] memory, uint256[] memory) {
    if (totalSupply() == 0) revert TotalSupplyZero();
    uint256 len = tokens.length;
    // ...
    for (uint256 i = 0; i < len; ) {
        // @audit isTokenRegistered always returns true at this condition, because all assets in tokens are registered.
        if (!isTokenRegistered(tokens[i])) {
            unchecked {
                ++i;
            }
            continue;
        }
        assets[cnt] = tokens[i];
        assetAmounts[cnt] = assetForVaultShares(vaultShare, tokens[i]);
        // ...
}
```

Recommendation(s): Remove the checking for `isTokenRegistered`.

Status: Fixed.

Update from the client: Fixed in commit 14bd2d01ae99501cb539a5765facba78ea66a675.

6.8 [Best Practice] Unused array in `_verifyPositionLimits(...)`

File(s): `src/LrtSquare.sol`

Description: The function `_verifyPositionLimits(...)` calculates the position weight and checks if it did not breach the token weight limit. The function uses an array for this checking unnecessary.

```
 function _verifyPositionLimits() internal view {
    uint256 len = tokens.length;
    uint64[] memory positionWeightLimits = new uint64[](len);
    //...

    for (uint256 i = 0; i < len; ) {
        // @audit it does not need an array for positionWeightLimits
        positionWeightLimits[i] = _getPositionWeight(tokens[i], vaultTotalValue);
        if (positionWeightLimits[i] > tokenInfos[tokens[i]].positionWeightLimit) revert TokenWeightLimitBreached(); {
            ++i;
        }
    }
}
```

Recommendation(s): Change the array `positionWeightLimits` for the type `uint64`.

Status: Fixed.

Update from the client: Fixed in commit 4d99bd03cb9261c4000e61420b891c4730312213.

6.9 [Best Practice] Unused event

File(s): src/LrtSquare.sol

Description: The event GovernorSet in contract LrtSquare is never used.

```
// @audit unused event
event GovernorSet(address oldGovernor, address newGovernor);
```

Recommendation(s): Consider removing the event if there is no intention to use it.

Status: Fixed.

Update from the client: Fixed in commit 4d99bd03cb9261c4000e61420b891c4730312213.

6.10 [Best Practice] swap function should implement reentrancy guard

File(s): src/Swapper1InchV6.sol

Description: The function swap(...) in the Swapper1InchV6 contract make a call with arbitrary data in _data as presented below. As a best practice, it is recommended using reentrancy guard in the function.

```
    function swap(
        address _fromAsset,
        address _toAsset,
        uint256 _fromAssetAmount,
        uint256 _minToAssetAmount,
        bytes calldata _data
    ) external returns (uint256 toAssetAmount) {
        if (IERC20(_fromAsset).allowance(address(this), swapRouter) < _fromAssetAmount)
            IERC20(_fromAsset).forceApprove(swapRouter, type(uint256).max);

        uint256 toAssetBalBefore = IERC20(_toAsset).balanceOf(msg.sender);

        (bool success, ) = swapRouter.call(_data);
        // ...
}
```

Recommendation(s): Add a nonReentrant modifier or a similar mechanism.

Status: Fixed.

Update from the client: Fixed in commit d356220ce021afaa97dafe33ed7507badc149ea2.

7 Documentation Evaluation

Software documentation refers to the written or visual information that describes the functionality, architecture, design, and implementation of software. It provides a comprehensive overview of the software system and helps users, developers, and stakeholders understand how the software works, how to use it, and how to maintain it. Software documentation can take different forms, such as user manuals, system manuals, technical specifications, requirements documents, design documents, and code comments. Software documentation is critical in software development, enabling effective communication between developers, testers, users, and other stakeholders. It helps to ensure that everyone involved in the development process has a shared understanding of the software system and its functionality. Moreover, software documentation can improve software maintenance by providing a clear and complete understanding of the software system, making it easier for developers to maintain, modify, and update the software over time. Smart contracts can use various types of software documentation. Some of the most common types include:

- Technical whitepaper: A technical whitepaper is a comprehensive document describing the smart contract's design and technical details. It includes information about the purpose of the contract, its architecture, its components, and how they interact with each other;
- User manual: A user manual is a document that provides information about how to use the smart contract. It includes step-by-step instructions on how to perform various tasks and explains the different features and functionalities of the contract;
- Code documentation: Code documentation is a document that provides details about the code of the smart contract. It includes information about the functions, variables, and classes used in the code, as well as explanations of how they work;
- API documentation: API documentation is a document that provides information about the API (Application Programming Interface) of the smart contract. It includes details about the methods, parameters, and responses that can be used to interact with the contract;
- Testing documentation: Testing documentation is a document that provides information about how the smart contract was tested. It includes details about the test cases that were used, the results of the tests, and any issues that were identified during testing;
- Audit documentation: Audit documentation includes reports, notes, and other materials related to the security audit of the smart contract. This type of documentation is critical in ensuring that the smart contract is secure and free from vulnerabilities.

These types of documentation are essential for smart contract development and maintenance. They help ensure that the contract is properly designed, implemented, and tested, and they provide a reference for developers who need to modify or maintain the contract in the future.

Remarks about the LRT Square documentation

The documentation for the LRT Square protocol was provided through a GitBook with high-level description. Moreover, the LRT Square team addressed all questions and concerns raised by the Nethermind Security team, providing valuable insights and a comprehensive understanding of the project's technical aspects.

8 Test Suite Evaluation

8.1 Compilation Output

```
> forge build
[] Compiling...
[] Compiling 120 files with Solc 0.8.25
[] Solc 0.8.25 finished in 177.43s
Compiler run successful!
```

8.2 Tests Output

```
\begin{minted}[]{shell}
  lrt-square git:(audit/nethermind) yarn test
yarn run v1.22.22
$ forge test
[] Compiling...
No files changed, compilation skipped

Ran 13 tests for test/LRTSquare/Pause.t.sol:LRTSquarePauseTest
[PASS] test_CanUnpauseAfterCommunityPause() (gas: 115472)
[PASS] test_CanUnpauseIfCommunityPauseDepositNotWithdrawn() (gas: 113140)
[PASS] test_CannotCommunityPauseIfDepositAmountNotSet() (gas: 19963)
[PASS] test_CannotCommunityPauseIfIncorrectDepositAmountIsSent() (gas: 54633)
[PASS] test_CannotCommunityPauseWhenAlreadyPaused() (gas: 83530)
[PASS] test_CannotDepositWhenPaused() (gas: 237799)
[PASS] test_CannotPauseWhenAlreadyPaused() (gas: 45325)
[PASS] test_CannotUnpauseWhenAlreadyPaused() (gas: 20203)
[PASS] test_CommunityPause() (gas: 108153)
[PASS] test_OnlyPauserCanPause() (gas: 20070)
[PASS] test_OnlyPauserCanUnpause() (gas: 49540)
[PASS] test_PauserCanPause() (gas: 44610)
[PASS] test_PauserCanUnpause() (gas: 36017)
Suite result: ok. 13 passed; 0 failed; 0 skipped; finished in 3.59ms (573.13µs CPU time)

Ran 2 tests for test/LRTSquare/UpdatePriceProvider.t.sol:LRTSquareUpdatePriceProviderTest
[PASS] test_CannotUpdatePriceProviderIfAddressZero() (gas: 388710)
[PASS] test_UpdatePriceProviderWithGovernance() (gas: 494266)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 3.50ms (1.07ms CPU time)

Ran 1 test for test/LRTSquare/AvsRewardsScenario.t.sol:LrtSquareTestAvsRewardScenario
[PASS] test_avs_rewards_scenario_1() (gas: 629379)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 4.63ms (1.18ms CPU time)

Ran 3 tests for test/LRTSquare/WhitelistToken.t.sol:LRTSquareWhitelistTokenTest
[PASS] test_CannotWhitelistTokenIfAddressZero() (gas: 327493)
[PASS] test_CannotWhitelistTokenIfNotAlreadyRegistered() (gas: 334074)
[PASS] test_WhitelistTokenWithGovernance() (gas: 329524)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 4.87ms (1.14ms CPU time)

Ran 3 tests for test/LRTSquare/SetDepositors.t.sol:LRTSquareSetDepositorsTest
[PASS] test_CannotSetDepositorIfAddressZero() (gas: 391863)
[PASS] test_CannotSetDepositorIfArrayLengthMismatch() (gas: 392724)
[PASS] test_SetDepositorsWithGovernance() (gas: 409745)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 5.02ms (868.71µs CPU time)

Ran 2 tests for test/LRTSquare/Redeem.t.sol:LRTSquareRedeemTest
[PASS] test_CannotRedeemIfInsufficientShares() (gas: 20157)
[PASS] test_Redeem() (gas: 170626)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 5.19ms (141.17µs CPU time)

Ran 7 tests for test/LRTSquare/Deposit.t.sol:LRTSquareDepositTest
[PASS] test_CannotDepositIfArrayLengthMismatch() (gas: 40000)
[PASS] test_CannotDepositIfPriceIsZero() (gas: 45790)
[PASS] test_CannotDepositIfRecipientAddress0() (gas: 37200)
[PASS] test_CannotDepositIfTokenNotRegistered() (gas: 53285)
[PASS] test_CannotDepositIfTokenNotWhitelisted() (gas: 389886)
[PASS] test_Deposit() (gas: 819381)
[PASS] test_OnlyDepositorsCanDeposit() (gas: 38865)
Suite result: ok. 7 passed; 0 failed; 0 skipped; finished in 5.30ms (1.59ms CPU time)

Ran 5 tests for test/LRTSquare/RegisterToken.t.sol:LRTSquareRegisterTokenTest
[PASS] test_CannotRegisterTokenIfAddressZero() (gas: 389515)
[PASS] test_CannotRegisterTokenIfAlreadyRegistered() (gas: 724912)
[PASS] test_CannotRegisterTokenIfMaxPercentageIsTooHigh() (gas: 404891)
[PASS] test_CannotRegisterTokenIfPriceNotConfigured() (gas: 404867)
[PASS] test_RegisterTokenWithGovernance() (gas: 465871)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 5.26ms (2.04ms CPU time)
```

```
Ran 8 tests for test/LRTSquare/Basics.t.sol:LRTSquareBasicsTest
[PASS] test_AssetOf() (gas: 882964)
[PASS] test_AssetsOf() (gas: 867756)
[PASS] test_CannotGetTokenValuesInEthIfArrayLengthMismatch() (gas: 27034)
[PASS] test_CannotGetTokenValuesInEthIfTokenNotRegistered() (gas: 17625)
[PASS] test_Deploy() (gas: 110890)
[PASS] test_PreviewDeposit() (gas: 904964)
[PASS] test_TokenValuesInEth() (gas: 64732)
[PASS] test_TotalAssetsValueInEth() (gas: 869742)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 5.84ms (4.08ms CPU time)

Ran 12 tests for test/LRTSquare/RateLimit.t.sol:LRTSquareRateLimitTest
[PASS] test_BucketRefills() (gas: 212336)
[PASS] test_CanChangePercentageRateLimit() (gas: 324332)
[PASS] test_CanChangeRateLimitConfig() (gas: 341631)
[PASS] test_CanChangeRateLimitRefillRate() (gas: 328465)
[PASS] test_CanChangeRateLimitTimePeriod() (gas: 324363)
[PASS] test_CanMintUpToRateLimit() (gas: 192571)
[PASS] test_CannotMintMoreThanRateLimit() (gas: 134927)
[PASS] test_CannotPrankTheRateLimit() (gas: 270691)
[PASS] test_OnlyGovernorCanChangePercentageRateLimit() (gas: 17300)
[PASS] test_OnlyGovernorCanChangeRateLimitConfig() (gas: 17511)
[PASS] test_OnlyGovernorCanChangeRateLimitRefillRate() (gas: 17321)
[PASS] test_OnlyGovernorCanSetRateLimitTimePeriod() (gas: 17290)
Suite result: ok. 12 passed; 0 failed; 0 skipped; finished in 5.63ms (2.62ms CPU time)

Ran 6 tests for test/LRTSquare/TokenLimits.t.sol:LRTSquareTokenLimitTest
[PASS] test_CanDepositUpToMaxPercentage() (gas: 818285)
[PASS] test_CannotDepositOverMaxPercentage() (gas: 775208)
[PASS] test_CannotSetMaxPercentageForATokenIfPercentageIsGreaterThanHundred() (gas: 333822)
[PASS] test_CannotSetMaxPercentageForATokenIfTokenIsAddressZero() (gas: 325932)
[PASS] test_CannotSetMaxPercentageForATokenIfTokenNotRegistered() (gas: 328189)
[PASS] test_SetMaxPercentageForATokenInVault() (gas: 330102)
Suite result: ok. 6 passed; 0 failed; 0 skipped; finished in 5.51ms (2.77ms CPU time)

Ran 5 tests for test/PriceProvider/PriceProvider.t.sol:PriceProviderTest
[PASS] test_BtcEthOracle() (gas: 51623)
[PASS] test_CanAddNewOracle() (gas: 95385)
[PASS] test_EthOracle() (gas: 11533)
[PASS] test_ExchangeRatePrice() (gas: 64087)
[PASS] test_MaticUsdOracle() (gas: 232379)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 6.81s (5.01s CPU time)

Ran 1 test for test/Swapper/Swapper1Inch.t.sol:Swapper1InchV6Test
[PASS] test_Swap() (gas: 391580)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 8.93s (4.85s CPU time)

Ran 13 tests for test/LRTSquare/Rebalance.t.sol:LRTSquareRebalanceTest
[PASS] test_CanRebalance() (gas: 772831)
[PASS] test_CannotRebalanceIfInputTokenNotRegistered() (gas: 22796)
[PASS] test_CannotRebalanceIfOutputTokenIsNotARegisteredValidOutputToken() (gas: 37418)
[PASS] test_CannotRebalanceIfOutputTokenNotRegistered() (gas: 24977)
[PASS] test_CannotRebalanceIfOutputTokenNotWhitelisted() (gas: 38169)
[PASS] test_CannotWhitelistAddressZeroAsRebalanceOutputToken() (gas: 18163)
[PASS] test_CannotWhitelistAsRebalanceOutputTokenIfPriceNotConfigured() (gas: 38568)
[PASS] test_CannotWhitelistAsRebalanceOutputTokenIfTokenNotRegistered() (gas: 20420)
[PASS] test_MaxSlippageCannotBeZero() (gas: 17959)
[PASS] test_OnlyGovernorCanSetRebalancer() (gas: 40498)
[PASS] test_OnlyGovernorCanWhitelistRebalanceOutputTokens() (gas: 112245)
[PASS] test_OnlyRebalancerCanSetMaxSlippage() (gas: 39499)
[PASS] test_RebalancerCannotBeAddressZero() (gas: 18032)
Suite result: ok. 13 passed; 0 failed; 0 skipped; finished in 9.18s (2.37s CPU time)

Ran 14 test suites in 9.18s (24.97s CPU time): 81 tests passed, 0 failed, 0 skipped (81 total tests)
  Done in 9.71s.
```

9 About Nethermind

Nethermind is a Blockchain Research and Software Engineering company. Our work touches every part of the web3 ecosystem - from layer 1 and layer 2 engineering, cryptography research, and security to application-layer protocol development. We offer strategic support to our institutional and enterprise partners across the blockchain, digital assets, and DeFi sectors, guiding them through all stages of the research and development process, from initial concepts to successful implementation.

We offer security audits of projects built on EVM-compatible chains and Starknet. We are active builders of the Starknet ecosystem, delivering a node implementation, a block explorer, a Solidity-to-Cairo transpiler, and formal verification tooling. Nethermind also provides strategic support to our institutional and enterprise partners in blockchain, digital assets, and decentralized finance (DeFi). In the next paragraphs, we introduce the company in more detail.

Blockchain Security: At Nethermind, we believe security is vital to the health and longevity of the entire Web3 ecosystem. We provide security services related to Smart Contract Audits, Formal Verification, and Real-Time Monitoring. Our Security Team comprises blockchain security experts in each field, often collaborating to produce comprehensive and robust security solutions. The team has a strong academic background, can apply state-of-the-art techniques, and is experienced in analyzing cutting-edge Solidity and Cairo smart contracts, such as ArgentX and StarkGate (the bridge connecting Ethereum and StarkNet). Most team members hold a Ph.D. degree and actively participate in the research community, accounting for 240+ articles published and 1,450+ citations in Google Scholar. The security team adopts customer-oriented and interactive processes where clients are involved in all stages of the work.

Blockchain Core Development: Our core engineering team, consisting of over 20 developers, maintains, improves, and upgrades our flagship product - the Nethermind Ethereum Execution Client. The client has been successfully operating for several years, supporting both the Ethereum Mainnet and its testnets, and now accounts for nearly a quarter of all synced Mainnet nodes. Our unwavering commitment to Ethereum's growth and stability extends to sidechains and layer 2 solutions. Notably, we were the sole execution layer client to facilitate Gnosis Chain's Merge, transitioning from Aura to Proof of Stake (PoS), and we are actively developing a full-node client to bolster Starknet's decentralization efforts. Our core team equips partners with tools for seamless node set-up, using generated docker-compose scripts tailored to their chosen execution client and preferred configurations for various network types.

DevOps and Infrastructure Management: Our infrastructure team ensures our partners' systems operate securely, reliably, and efficiently. We provide infrastructure design, deployment, monitoring, maintenance, and troubleshooting support, allowing you to focus on your core business operations. Boasting extensive expertise in Blockchain as a Service, private blockchain implementations, and node management, our infrastructure and DevOps engineers are proficient with major cloud solution providers and can host applications in-house or on clients' premises. Our global in-house SRE teams offer 24/7 monitoring and alerts for both infrastructure and application levels. We manage over 5,000 public and private validators and maintain nodes on major public blockchains such as Polygon, Gnosis, Solana, Cosmos, Near, Avalanche, Polkadot, Aptos, and StarkWare L2. Sedge is an open-source tool developed by our infrastructure experts, designed to simplify the complex process of setting up a proof-of-stake (PoS) network or chain validator. Sedge generates docker-compose scripts for the entire validator set-up based on the chosen client, making the process easier and quicker while following best practices to avoid downtime and being slashed.

Cryptography Research: At Nethermind, our Cryptography Research team is dedicated to continuous internal research while fostering close collaboration with external partners. The team has expertise across a wide range of domains, including cryptography protocols, consensus design, decentralized identity, verifiable credentials, Sybil resistance, oracles, and credentials, distributed validator technology (DVT), and Zero-knowledge proofs. This diverse skill set, combined with strong collaboration between our engineering teams, enables us to deliver cutting-edge solutions to our partners and clients.

Smart Contract Development & DeFi Research: Our smart contract development and DeFi research team comprises 40+ world-class engineers who collaborate closely with partners to identify needs and work on value-adding projects. The team specializes in Solidity and Cairo development, architecture design, and DeFi solutions, including DEXs, AMMs, structured products, derivatives, and money market protocols, as well as ERC20, 721, and 1155 token design. Our research and data analytics focuses on three key areas: technical due diligence, market research, and DeFi research. Utilizing a data-driven approach, we offer in-depth insights and outlooks on various industry themes.

Our suite of L2 tooling: Warp is Starknet's approach to EVM compatibility. It allows developers to take their Solidity smart contracts and transpile them to Cairo, Starknet's smart contract language. In the short time since its inception, the project has accomplished many achievements, including successfully transpiling Uniswap v3 onto Starknet using Warp.

- **Voyager** is a user-friendly Starknet block explorer that offers comprehensive insights into the Starknet network. With its intuitive interface and powerful features, Voyager allows users to easily search for and examine transactions, addresses, and contract details. As an essential tool for navigating the Starknet ecosystem, Voyager is the go-to solution for users seeking in-depth information and analysis;
- **Horus** is an open-source formal verification tool for StarkNet smart contracts. It simplifies the process of formally verifying Starknet smart contracts, allowing developers to express various assertions about the behavior of their code using a simple assertion language;
- **Juno** is a full-node client implementation for Starknet, drawing on the expertise gained from developing the Nethermind Client. Written in Golang and open-sourced from the outset, Juno verifies the validity of the data received from Starknet by comparing it to proofs retrieved from Ethereum, thus maintaining the integrity and security of the entire ecosystem.

Learn more about us at nethermind.io.

General Advisory to Clients

As auditors, we recommend that any changes or updates made to the audited codebase undergo a re-audit or security review to address potential vulnerabilities or risks introduced by the modifications. By conducting a re-audit or security review of the modified codebase, you can significantly enhance the overall security of your system and reduce the likelihood of exploitation. However, we do not possess the authority or right to impose obligations or restrictions on our clients regarding codebase updates, modifications, or subsequent audits. Accordingly, the decision to seek a re-audit or security review lies solely with you.

Disclaimer