OMNISCIA

March 18, 2025

SMART CONTRACT AUDIT REPORT

Lair Finance
Bughole Restaking

omniscia.io

info@omniscia.io

Online report: lair-finance-bughole-restaking

Omniscia.io is one of the fastest growing and most trusted blockchain security firms and has rapidly become a true market leader. To date, our team has collectively secured over 370+ clients, detecting 1,500+ high-severity issues in widely adopted smart contracts.

Founded in France at the start of 2020, and with a track record spanning back to 2017, our team has been at the forefront of auditing smart contracts, providing expert analysis and identifying potential vulnerabilities to ensure the highest level of security of popular smart contracts, as well as complex and sophisticated decentralized protocols.

Our clients, ecosystem partners, and backers include leading ecosystem players such as L'Oréal, Polygon, AvaLabs, Gnosis, Morpho, Vesta, Gravita, Olympus DAO, Fetch.ai, and LimitBreak, among others.

To keep up to date with all the latest news and announcements follow us on twitter @omniscia_sec.


omniscia.io


info@omniscia.io

Online report: lair-finance-bughole-restaking

Bughole Restaking Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
283dc2cdee	February 24th 2025	6a5252defc
09c58d1f17	March 6th 2025	e2d3368143
803a3419a9	March 17th 2025	f343cd21df
803a3419a9	March 17th 2025	bdad1ffe1f
803a3419a9	March 18th 2025	f0da8ca41e

Audit Overview

We were tasked with performing an audit of the Lair Finance codebase and in particular their Bughole Restaking module.

Over the course of the audit, we identified a mechanism via which the pair token amount claims can be repeated infinitely and thus siphon funds from the system.

The system appears to be in a mid-development state and contains several redundant code statements as well as inefficient code.

While we have identified several optimizations that can be applied, the list is not exhaustive and we strongly recommend a refactor of the codebase to be performed so as to limit its attack surface and optimize its legibility.

Finally, the `ReStakingManager` implementation interacts with several out-of-scope contracts. We were able to observe a single potential issue based on a localized inconsistency within the `ReStakingManager`, however, we advise the Lair Finance team to perform an audit of the relevant externally integrated contracts to ensure that the integration performed within the `ReStakingManager` is correct as this point has not been validated as part of the audit.

We advise the Lair Finance team to closely evaluate all minor-and-above findings identified in the report and promptly remediate them as well as consider all optimizational exhibits identified in the report.

Post-Audit Conclusion

The Lair Finance team iterated through all findings within the report and provided us with a revised commit hash to evaluate all exhibits on.

We evaluated all alleviations performed by Lair Finance and have identified that certain exhibits have not been adequately dealt with. We advise the Lair Finance team to revisit the following exhibits: `RST-02M`, `RST-01M`, `RSM-09M`, `RSM-02M`, `RSM-01M`, `RSM-01S`

Most of the optimizations have been acknowledged, however, we would like to note that we introduced a follow-up optimization on the alleviation chapter of the `RSM-04M` exhibit for the Lair Finance team's consumption.

Post-Audit Conclusion (803a3419a9)

The Lair Finance team evaluated our follow-up recommendations as well as open exhibits and proceeded to alleviate them.

In addition to the above, the Lair Finance team engaged in extensive discussions with our team regarding various concerns and misbehaviors they were identifying as we were progressing through the audit.

Specifically, they identified the following issues:

- Updates of the pair rate would result in a `revert` beyond the reward period
- Unstake operations were not evaluating the available funds to extract

We advised the Lair Finance team how to optimally address those two issues, and we observed and shared a specification discrepancy in the implementation of the **`ReStakingManager::setTotalPairTokenAmount`** function.

Additionally, we observed certain optimizations that could be applied to the codebase in the interim commit hashes that included the Lair Finance team's fixes.

The Lair Finance team proceeded to address the aforementioned issues as well as the design discrepancy via the following actions:

- Returning early during pair rate updates if the time delta is `0`
- Adding the `totalPairTokenStakedAmount` to the available pair token funds for withdrawals
- Reverting on total pair token amount reconfigurations if the `pairEndTime` is in the future (i.e. rewards are ongoing)

We consider all remediations to have been properly applied throughout the codebase thus far, however, we have observed a potential design discrepancy in the codebase.

The creation of an unstake will calculate the `nativeTokenAmount` that the `reStakingTokenAmount` corresponds to and will burn the `reStakingTokenAmount`.

The claim of an unstake will utilize the `reStakingTokenAmount` when performing a withdrawal through the treasury whilst it should (presumably) utilize the `nativeTokenAmount`.

The Lair Finance team was unable to provide us with documentation on the aforementioned point, so we would like to invite them to evaluate whether this is intended behaviour.

Post-Audit Conclusion (Follow-Up)

The Lair Finance team evaluated the issue we outlined in the previous chapter and clarified that it is intended behavior.

Specifically, the denomination as well as exchange rate between the re-staking token implementation and the staking token implementation is `1:1`, permitting the same value to be utilized for burning and withdrawing funds from the treasury.

We consider all items of the audit report to have been properly evaluated by the Lair Finance team with no outstanding remediative actions remaining based on the audit scope.

As mentioned in the original audit report summary, we advise the Lair Finance team to closely evaluate the integration points of the code with out-of-scope code, such as the conversions of tokens, transfer address exchanges, and any other contract not explicitly covered, to ensure that they behave as expected.

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	0	0	0	0
Informational	20	2	0	18
Minor	3	1	0	2
Medium	5	5	0	0
Major	1	1	0	0

During the audit, we filtered and validated a total of **2 findings utilizing static analysis** tools as well as identified a total of **27 findings during the manual review** of the codebase. We strongly recommend that any minor severity or higher findings are dealt with promptly prior to the project's launch as they can introduce potential misbehaviours of the system as well as exploits.

- 🎯 **Scope**
- 💻 **Compilation**
- 🔍 **Static Analysis**
- 👁️ **Manual Review**
- 🖌️ **Code Style**

Scope

The audit engagement encompassed a specific list of contracts that were present in the commit hash of the repository that was in scope. The tables below detail certain meta-data about the target of the security assessment and a navigation chart is present at the end that links to the relevant findings per file.

Target

- Repository: **https://github.com/bug4city/bughole-lsd**
- Commit: 283dc2cdee1f378d0cd01b3341d5047571d7cabf
- Language: Solidity
- Network: Kaia
- Revisions: **283dc2cdee**, **09c58d1f17**, **4599c0736a**, **803a3419a9**

Contracts Assessed

File	Total Finding(s)
contracts/restaking/ReStakingManager/ReStakingManager.sol (RSM)	24
contracts/restaking/Treasury/ReStakingTreasury.sol (RST)	5
contracts/restaking/structs/Unstake.sol (UEK)	0

Compilation

The project utilizes `hardhat` as its development pipeline tool, containing an array of tests and scripts coded in TypeScript.

To compile the project, the `compile` command needs to be issued via the `npx` CLI tool to `hardhat`:

```bash
npx hardhat compile
```

The `hardhat` tool automatically selects Solidity version `0.8.25` based on the version specified within the `hardhat.config.ts` file.

The project contains discrepancies with regards to the Solidity version used as the `pragma` statements of the contracts are open-ended (`^0.8.20`).

We advise them to be locked to `0.8.25` (`=0.8.25`), the same version utilized for our static analysis as well as optimizational review of the codebase.

During compilation with the `hardhat` pipeline, no errors were identified that relate to the syntax or bytecode size of the contracts.

Static Analysis

The execution of our static analysis toolkit identified **61 potential issues** within the codebase of which **50 were ruled out to be false positives** or negligible findings.

The remaining **11 issues** were validated and grouped and formalized into the **2 exhibits** that follow:

ID	Severity	Addressed	Title
RSM-01S	Informational	Acknowledged	Illegible Numeric Value Representations
RSM-02S	Informational	Yes	Inexistent Event Emissions

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Lair Finance's restaking system.

As the project at hand implements a specialized staking system, intricate care was put into ensuring that the **flow of funds within the system conforms to the specifications and restrictions** laid forth within the protocol's specification.

We validated that **all state transitions of the system occur within sane criteria** and that all rudimentary formulas within the system execute as expected. We **pinpointed a significant fund-affecting vulnerability** within the system among other moderate issues all of which could have had **severe ramifications** to its overall operation; for more information, kindly consult the relevant major and medium severity findings within the audit report.

Additionally, the system was investigated for any other commonly present attack vectors such as re-entrancy attacks, mathematical truncations, logical flaws and **ERC / EIP** standard inconsistencies. The documentation of the project was satisfactory to the extent it need be, however, the system's out-of-scope integrations could not be validated as part of the audit engagement.

A total of **27 findings** were identified over the course of the manual review of which **11 findings** concerned the behaviour and security of the system. The non-security related findings, such as optimizations, are included in the separate **Code Style** chapter.

The finding table below enumerates all these security / behavioural findings:

ID	Severity	Addressed	Title
RSM-01M	Informational	Acknowledged	Inexistent Override of Role Grant
RSM-02M	Minor	Acknowledged	Improper Role Removal Restriction
RSM-03M	Minor	Yes	Incorrect Restriction
RSM-04M	Medium	Nullified	Discrepant Evaluation of Staking Token
RSM-05M	Medium	Yes	Incorrect Low-Level Interactions

RSM-06M	Medium	Yes	Incorrect Sequence of Reward Distribution
RSM-07M	Medium	Yes	Inexistent Slippage Control Variable of Pair Token Amount
RSM-08M	Medium	Yes	Insecure Updates of Reward Times
RSM-09M	Major	Nullified	Inexistent Deletion of Claimed Pair Token Amount
RST-01M	Informational	Acknowledged	Inexistent Override of Role Grant
RST-02M	Minor	Acknowledged	Improper Role Removal Restriction

Code Style

During the manual portion of the audit, we identified **16 optimizations** that can be applied to the codebase that will decrease the operational cost associated with the execution of a particular function and generally ensure that the project complies with the latest best practices and standards in Solidity.

Additionally, this section of the audit contains any opinionated adjustments we believe the code should make to make it more legible as well as truer to its purpose.

These optimizations are enumerated below:

ID	Severity	Addressed	Title
RSM-01C	Informational	Acknowledged	Deprecated Revert Pattern
RSM-02C	Informational	Acknowledged	Ineffectual Usage of Safe Arithmetics
RSM-03C	Informational	Acknowledged	Inefficient Initialization Flow
RSM-04C	Informational	Yes	Inefficient Loop Limit Evaluation
RSM-05C	Informational	Acknowledged	Inefficient Variable Mutabilities
RSM-06C	Informational	Acknowledged	Inefficient `mapping` Lookups
RSM-07C	Informational	Acknowledged	Redundant Application of Modifiers
RSM-08C	Informational	Acknowledged	Redundant Balance & Allowance Measurements
RSM-09C	Informational	Acknowledged	Redundant Storage Reads
RSM-10C	Informational	Acknowledged	Redundant Zero-Value Assignments

RSM-11C	Informational	Acknowledged	Repetitive Value Literals
RSM-12C	Informational	Acknowledged	Sub-Optimal Early Return
RSM-13C	Informational	Acknowledged	Suboptimal Struct Declaration Styles
RST-01C	Informational	Acknowledged	Deprecated Revert Pattern
RST-02C	Informational	Acknowledged	Inefficient Balance Evaluation
RST-03C	Informational	Acknowledged	Inefficient Initialization Flow

ReStakingManager Static Analysis Findings

RSM-01S: Illegible Numeric Value Representations

Type	Severity	Location
Code Style	Informational	**ReStakingManager.sol**: • I-1: **L65** • I-2: **L151** • I-3: **L509** • I-4: **L549**

Description:

The linked representations of numeric literals are sub-optimally represented decreasing the legibility of the codebase.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```sol
65  uint256 private constant DECIMALS = 10**18;
```

Recommendation:

To properly illustrate each value's purpose, we advise the following guidelines to be followed. For values meant to depict fractions with a base of `1e18`, we advise fractions to be utilized directly (i.e. `1e17` becomes `0.1e18`) as they are supported. For values meant to represent a percentage base, we advise each value to utilize the underscore (`_`) separator to discern the percentage decimal (i.e. `10000` becomes `100_00`, `300` becomes `3_00` and so on). Finally, for large numeric values we simply advise the underscore character to be utilized again to represent them (i.e. `1000000` becomes `1_000_000`).

Alleviation (09c58d1f17):

The Lair Finance team proceeded with addressing solely the first of the four highlighted instances and did so in a non-standard way.

We advise `10**18` to be represented by either `1 ether` or `1e18` and the remaining instances to be adjusted as advised.

Alleviation (803a3419a9):

The Lair Finance team evaluated our follow-up recommendation and opted to retain the codebase as-is, rendering the exhibit to be ultimately acknowledged.

RSM-02S: Inexistent Event Emissions

Type	Severity	Location
Language Specific	Informational	**ReStakingManager.sol**: • I-1: **L171-L173** • I-2: **L187-L189** • I-3: **L203-L205** • I-4: **L219-L221** • I-5: **L235-L237** • I-6: **L251-L255** • I-7: **L277-L279**

Description:

The linked functions adjust sensitive contract variables yet do not emit an event for it.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```sol
171 function setStakingNodeAddress(address _stakingNodeAddress) public
validAddress(_stakingNodeAddress) onlyRole(DEFAULT_ADMIN_ROLE) {
172     stakingNodeAddress = _stakingNodeAddress;
173 }
```

Recommendation:

We advise an `event` to be declared and correspondingly emitted for each function to ensure off-chain processes can properly react to this system adjustment.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The `UpdateStakingNode`, `UpdateStakingToken`, `UpdateReStakingToken`, `UpdateTreasury`, `UpdateTransfer`, `UpdateClaimIntervalTime`, and `UpdateStakingNodeController` events were introduced to the codebase and are correspondingly emitted in the **`ReStakingManager::setStakingNodeAddress`**, **`ReStakingManager::setStakingTokenAddress`**, **`ReStakingManager::setReStakingTokenAddress`**, **`ReStakingManager::setTreasuryAddress`**, **`ReStakingManager::setTransferAddress`**, and **`ReStakingManager::setStakingNodeControllerAddress`** functions respectively, addressing this exhibit in full.

ReStakingManager Manual Review Findings

RSM-01M: Inexistent Override of Role Grant

Type	Severity	Location
Logical Fault	Informational	ReStakingManager.sol:L845-L847, L854-L856

Description:

The `AccessControlUpgradeable::grantRole` function is not overridden by the `ReStakingManager` implementation even though other functions are.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```sol
840 /**
841    * @dev grantRole
842    * @param role role
843    * @param account account
844    */
845 function revokeRole(bytes32 role, address account) public override
onlyRole(getRoleAdmin(role)) {
846     revert("ReStakingManager:: revokeRole is disabled");
847 }
848
849 /**
```

Example (Cont.):

```sol
    * @dev renounceRole
    * @param role role
    * @param account account
    */
function renounceRole(bytes32 role, address account) public override {
    revert("ReStakingManager:: renounceRole is disabled");
}
```

Recommendation:

We advise this to be clarified as intentional within the code or the function to be overridden correctly, either of which we consider an adequate resolution to this exhibit.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The Lair Finance team evaluated this exhibit and clarified that they do not envision any issue arising from not overriding the function, and have opted to not do so.

RSM-02M: Improper Role Removal Restriction

Type	Severity	Location
Logical Fault	Minor	ReStakingManager.sol:L804

Description:

The `ReStakingManager::removeRole` function is meant to disallow a `DEFAULT_ADMIN_ROLE` from revoking themselves as an administrator, however, the security check is improperly defined.

Specifically, it will evaluate whether the `account` has the `DEFAULT_ADMIN_ROLE` and is the caller but it will not evaluate that the `role` being removed is indeed the `DEFAULT_ADMIN_ROLE`.

Impact:

An administrator who attempts to remove a role for themselves will be unable to do so.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```sol
797 /**
798  * @dev remove role.
799   * @param role The name of the role by keccak256.
800   * @param account The address of the account.
801   */
802 function removeRole(bytes32 role, address account) public
onlyRole(DEFAULT_ADMIN_ROLE) {
803     require(
804         !(hasRole(DEFAULT_ADMIN_ROLE, account) && account == msg.sender),
805         "Admin cannot revoke own admin role"
806     );
```

Example (Cont.):

```sol

807
808     _revokeRole(role, account);
809 }
```

Recommendation:

We advise the security check to be corrected, permitting other roles to be self-removed as expected.

Alleviation (09c58d1f17):

The Lair Finance team evaluated this exhibit and stated that one administrator has to remain which is irrelevant to the restriction applied.

If the `role` being removed is not the `DEFAULT_ADMIN_ROLE`, the restriction is unnecessarily imposed rendering the exhibit to remain open.

Alleviation (803a3419a9):

The Lair Finance team evaluated our follow-up recommendation and opted to retain the codebase as-is, rendering the exhibit to be ultimately acknowledged.

RSM-03M: Incorrect Restriction

Type	Severity	Location
Logical Fault	Minor	ReStakingManager.sol:L888

Description:

The referenced restriction is incorrect as it will evaluate that the newly configured total pair amount is at least equal to the total pair token stored amount which represents the distributed value.

As such, it permits the actual `totalPairTokenStoredAmount` calculated by the function's execution to result in a lesser value and thus to a lower `pairRate`.

Impact:

It is presently possible to reconfigure the `totalPairTokenAmount` of the contract to a value that would reduce its rate and cause it to misbehave.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```sol
885 function setTotalPairTokenAmount(uint256 _totalPairTokenAmount) public
onlyRole(DEFAULT_ADMIN_ROLE) {
886     updatePairRate();
887
888     if (_totalPairTokenAmount < totalPairTokenStoredAmount) {
889         revert("ReStakingManager:: totalPairTokenAmount is less than
totalPairTokenStoredAmount");
890     }
891
892     totalPairTokenAmount = _totalPairTokenAmount;
893
894     updatePairRate();
```

Example (Cont.):

```sol
895 }
```

Recommendation:

We advise the code to ensure that the post-rate-update `totalPairTokenStoredAmount` is at least equal to or greater than the previous `totalPairTokenStoredAmount`, ensuring an adjustment can only result in an increase of the stored `pairRate`.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The code was updated to impose the limitation advised, ensuring that the `pairRate` of the system cannot lower.

RSM-04M: Discrepant Evaluation of Staking Token

Type	Severity	Location
Logical Fault	Medium	ReStakingManager.sol:L763, L767, L771, L775, L785

Description:

The token approval mechanism of the `ReStakingManager` indicates that any token can be introduced to the system, however, the `ReStakingManager::calculateStakingToken` implementation will solely support four different tokens and will **yield incorrect values for any other**.

Impact:

The `ReStakingManager::calculateStakingToken` function will yield `0` for any non-zero `tokenAmount` of an approved token that does not have an explicit `if` clause defined within it which we consider incorrect.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```sol
759 function calculateStakingToken(address tokenAddress, uint256 tokenAmount) public
view validAddress(tokenAddress) validMoreThanZero(tokenAmount) returns (uint256) {
760     checkApprovedToken(tokenAddress);
761
762     uint256 amount = 0;
763     if (tokenAddress == getStakingTokenAddress()) {
764         amount = tokenAmount;
765     }
766
767     if (tokenAddress == gcKlayAddress) { // gcKLAY = Kaia 1:1
768         amount =
IStakingToken(getStakingTokenAddress()).getRatioStakingTokenByNativeToken(tokenAmount
);
```

Example (Cont.):

```sol
    }

    if (tokenAddress == stKlayAddress) { // stKLAY = Kaia 1:1
        amount = IStakingToken(getStakingTokenAddress()).getRatioStakingTokenByNativeToken(tokenAmount);
    }

    if (tokenAddress == sKlayAddress) { // sKLAY = Kaia 1:n
        ICnStakingV2 cnStakingV2 = ICnStakingV2(sKlayGCAddress);
        IERC20 sKLAY = IERC20(tokenAddress);

        uint256 sKlayRatio = (cnStakingV2.staking() - cnStakingV2.unstaking()) * 10 ** 18 / sKLAY.totalSupply();
        uint256 nativeTokenAmount = tokenAmount * sKlayRatio / 10 ** 18;

        amount = IStakingToken(getStakingTokenAddress()).getRatioStakingTokenByNativeToken(nativeTokenAmount);
    }

    return amount;
}
```

Recommendation:

We advise the code of the `ReStakingManager::calculateStakingToken` to `revert` if a `tokenAddress` is not matched in its `if` clauses, or the `ReStakingManager::approveToken` function to solely permit the tokens that are explicitly supported by the system, either of which we consider an adequate alleviation to this exhibit.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The Lair Finance team clarified that the `ReStakingManager::checkApprovedToken` function ensures that the token evaluated is one of the permitted values, preventing any discrepancy from arising and thus invalidating this exhibit.

We would like to note that in this case, the code can be optimized by chaining the `if` conditionals and evaluating the last case as a simple `else` clause.

RSM-05M: Incorrect Low-Level Interactions

Type	Severity	Location
Logical Fault	Medium	**ReStakingManager.sol**: • I-1: **L736** • I-2: **L741**

Description:

The referenced low-level interactions are incorrect as they do not validate that they have been successfully executed.

Impact:

A failure in the `INodeManager::distributeReward` or `IOzys::refreshStaking` implementations will not cascade to a failure in the `ReStakingManager`, leading to potentially desynchronized states.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```sol
734 if (tokenAddress == stKlayAddress) {
735     //Stakely NodeManager
736 address(0x3F2F52FdDc40085b6a72CA955BCbf3Ba9611496d).call(abi.encodeWithSignature("distri
buteReward()"));
737 }
738
739 if (tokenAddress == sKlayAddress) {
740     //0x77777779eE2d933dA027Ee1fB3590c41529046c8 ozys
741 address(0x77777779eE2d933dA027Ee1fB3590c41529046c8).call(abi.encodeWithSignature("refres
hStaking()"));
742 }
```

Recommendation:

We advise the code to utilize an `interface` declaration for each invocation, permitting each function to be invoked using normal Solidity syntax and thus to validate successful execution automatically.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The code was updated to perform an `interface` based invocation on each instance, ensuring that failures properly bubble up and alleviating this exhibit.

RSM-06M: Incorrect Sequence of Reward Distribution

Type	Severity	Location
Logical Fault	Medium	ReStakingManager.sol:L646, L648

Description:

Although the `INodeController` implementation is out-of-scope of this audit engagement, its usage appears to be non-uniform as a stake operation will distribute rewards prior to any balance changes whereas the `ReStakingManager::_unstake` function will distribute rewards after assets have been burned.

Impact:

Rewards owed toward the `sender` will not be disbursed properly whenever an unstake occurs.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```
646 IReStakingToken(getReStakingTokenAddress()).burn(sender, reStakingTokenAmount);
647
648 INodeController(payable(getStakingNodeControllerAddress())).distributeReward();
```

Recommendation:

We consider this behaviour to be incorrect as most reward distribution systems will utilize balance measurements, and we advise the distribution of rewards to occur prior to the burn of the re-staking token amount.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The code was re-ordered per our recommendation, ensuring that the distribution of rewards occurs prior to the burn of any balance.

RSM-07M: Inexistent Slippage Control Variable of Pair Token Amount

Type	Severity	Location
Logical Fault	Medium	ReStakingManager.sol:L631, L643

Description:

An unstake operation cannot control the `pairTokenRealAmount` that the user is willing to accept for their unstake operation rendering them susceptible to slippage.

Impact:

A user performing an unstake operation via the **`ReStakingManager::unstake`** function will be unable to control the amount of funds they are willing to accept in the pair token denomination.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```sol
631 function _unstake(address sender, uint256 reStakingTokenAmount) private
validAddress(sender) validMoreThanZero(reStakingTokenAmount) returns (uint256,
uint256) {
632     require(getStakingTokenAddress() != address(0), "ReStakingManager::
stakingTokenAddress zero address");
633     require(getReStakingTokenAddress() != address(0), "ReStakingManager::
reStakingTokenAddress zero address");
634     require(getTreasuryAddress() != address(0), "ReStakingManager::
treasuryAddress zero address");
635     require(getTransferAddress() != address(0), "ReStakingManager::
transferAddress zero address");
636     require(IReStakingToken(getReStakingTokenAddress()).balanceOf(sender) >=
reStakingTokenAmount, "ReStakingManager:: amount is not enough");
637
638     uint256 pairTokenRealAmount = 0;
639
640     if (pairTokenAddress != address(0) && pairRate > 0) {
```

Example (Cont.):

```sol
641         uint256 pairAmount = pairRate.mul(reStakingTokenAmount).div(DECIMALS);
642         uint256 pairTokenRemainAmount =
totalPairTokenAmount.sub(totalPairTokenUnstakedAmount);
643         pairTokenRealAmount = pairAmount > pairTokenRemainAmount ?
pairTokenRemainAmount : pairAmount;
644     }
```

Recommendation:

We advise the code to introduce a slippage control variable, permitting the user to specify the minimum `pairTokenRealAmount` they are willing to accept for their unstake operation.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

A proper slippage control variable was introduced to the unstaking mechanism of the system, permitting users to control the amount of pair tokens they receive.

RSM-08M: Insecure Updates of Reward Times

Type	Severity	Location
Logical Fault	Medium	**ReStakingManager.sol**: • I-1: **L859-L869** • I-2: **L871-L879**

Description:

The referenced functions will update the start and end times of the reward period within the `ReStakingManager` which can result in the `pairRate` being reduced and thus the contract misbehaving.

Impact:

Any update of the start or end times of the reward distribution period while it is active will result in miscalculation of values and a reduction of the `pairRate`.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```sol
859 function setPairEndTime(uint256 _pairEndTime) public onlyRole(DEFAULT_ADMIN_ROLE)
{
860     if (_pairEndTime < pairStartTime) {
861         revert("ReStakingManager:: pairEndTime is less than pairStartTime");
862     }
863
864     updatePairRate();
865
866     pairEndTime = _pairEndTime;
867
868     updatePairRate();
```

Example (Cont.):

```sol
869 }
870
871 function setPairStartTime(uint256 _pairStartTime) public
onlyRole(DEFAULT_ADMIN_ROLE) {
872     if (pairLastUpdateTime < _pairStartTime) {
873         pairLastUpdateTime = _pairStartTime;
874     }
875
876     pairStartTime = _pairStartTime;
877
878     updatePairRate();
879 }
```

Recommendation:

We advise the code to prevent these time updates, or to introduce a proper "migration" of rewards by resetting the total pair token amount so as to ensure the reward distribution smoothly continues from that point onward.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The code was revised to ensure that time-based updates can solely occur when the system's pair rate has not been established yet; a restriction that we consider to be an adequate alleviation.

RSM-09M: Inexistent Deletion of Claimed Pair Token Amount

Type	Severity	Location
Logical Fault	Major	ReStakingManager.sol:L652-L656, L711

Description:

The `ReStakingManager::_claim` function will not erase the `pairTokenAmountList` data entry when processing it, permitting it to be reclaimed if a new unstake operation occurs over the same unstake list `index` that does not result in a `pairTokenRealAmount` that is non-zero (f.e., a claim of `1` wei of `reStakingTOkenAmount`).

Impact:

It is possible to re-claim the same pair token amount multiple times by ensuring the conditional write in the `ReStakingManager::_stake` function is not triggered.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```
710 if (pairTokenAmountList[sender][index] > 0) {
711     uint256 claimedPairTokenAmount = pairTokenAmountList[sender][index];
712     totalPairTokenClaimAmount =
totalPairTokenClaimAmount.add(claimedPairTokenAmount);
713     IReStakingTreasury(getTreasuryAddress()).withdraw(sender,
claimedPairTokenAmount, pairTokenAddress);
714 }
```

Recommendation:

We advise the code to properly erase the data entry prior to claiming it, preventing it to be recaptured multiple times incorrectly.

Alleviation (09c58d1f17):

The Lair Finance team did not introduce an alleviation for this exhibit and cited an improper restriction as limiting this operation.

As our original exhibit outlines, an unstake on the same index can occur and the conditional that would overwrite the `pairTokenAmountList` entry for a consequent unstake on the same index is triggered solely with a non-zero `pairTokenRealAmount`.

As that value is the result of a multiplication and division, a miniscule withdrawal (i.e. `1 wei`) can result in a claim operation being repeated on the same index without fresh `pairTokenAmountList` values.

Alleviation (803a3419a9):

After discussing this particular exhibit with the Lair Finance team, we concluded that it is inapplicable as indexes are never re-used within the system.

As such, we consider this exhibit to be nullified.

ReStakingTreasury Manual Review Findings

RST-01M: Inexistent Override of Role Grant

Type	Severity	Location
Logical Fault	Informational	ReStakingTreasury.sol:L163-L165, L172-L174

Description:

The `AccessControlUpgradeable::grantRole` function is not overridden by the `ReStakingTreasury` implementation even though other functions are.

Example:

contracts/restaking/Treasury/ReStakingTreasury.sol

SOL

```sol
158 /**
159   * @dev grantRole
160   * @param role role
161   * @param account account
162   */
163 function revokeRole(bytes32 role, address account) public override
onlyRole(getRoleAdmin(role)) {
164     revert("ReStakingTreasury:: revokeRole is disabled");
165 }
166
167 /**
```

Example (Cont.):

```sol
168   * @dev renounceRole
169   * @param role role
170   * @param account account
171   */
172 function renounceRole(bytes32 role, address account) public override {
173     revert("ReStakingTreasury:: renounceRole is disabled");
174 }
```

Recommendation:

We advise this to be clarified as intentional within the code or the function to be overridden correctly, either of which we consider an adequate resolution to this exhibit.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The Lair Finance team evaluated this exhibit and clarified that they do not envision any issue arising from not overriding the function, and have opted to not do so.

RST-02M: Improper Role Removal Restriction

Type	Severity	Location
Logical Fault	Minor	ReStakingTreasury.sol:L122

Description:

The `ReStakingTreasury::removeRole` function is meant to disallow a `DEFAULT_ADMIN_ROLE` from revoking themselves as an administrator, however, the security check is improperly defined.

Specifically, it will evaluate whether the `account` has the `DEFAULT_ADMIN_ROLE` and is the caller but it will not evaluate that the `role` being removed is indeed the `DEFAULT_ADMIN_ROLE`.

Impact:

An administrator who attempts to remove a role for themselves will be unable to do so.

Example:

contracts/restaking/Treasury/ReStakingTreasury.sol

SOL

```sol
120 function removeRole(bytes32 role, address account) public
onlyRole(DEFAULT_ADMIN_ROLE) {
121     require(
122         !(hasRole(DEFAULT_ADMIN_ROLE, account) && account == msg.sender),
123         "Admin cannot revoke own admin role"
124     );
125
126     _revokeRole(role, account);
127 }
```

Recommendation:

We advise the security check to be corrected, permitting other roles to be self-removed as expected.

Alleviation (09c58d1f17):

The Lair Finance team evaluated this exhibit and stated that one administrator has to remain which is irrelevant to the restriction applied.

If the `role` being removed is not the `DEFAULT_ADMIN_ROLE`, the restriction is unnecessarily imposed rendering the exhibit to remain open.

Alleviation (803a3419a9):

The Lair Finance team evaluated our follow-up recommendation and opted to retain the codebase as-is, rendering the exhibit to be ultimately acknowledged.

ReStakingManager Code Style Findings

RSM-01C: Deprecated Revert Pattern

Type	Severity	Location
Code Style	Informational	**ReStakingManager.sol**: • I-1: **L829** • I-2: **L837** • I-3: **L846** • I-4: **L855** • I-5: **L861** • I-6: **L889**

Description:

The referenced statements will issue a `revert` with a textual message rather than an `error` which has been deprecated.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```sol
829 revert("ReStakingManager:: ownership cannot be renounced");
```

Recommendation:

We advise a proper `error` to be declared for each instance and to be emitted, optimizing the code's syntax.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The Lair Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

RSM-02C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	ReStakingManager.sol:L437, L440-L447

Description:

The linked mathematical operations are guaranteed to be performed safely by surrounding conditionals evaluated in either `require` checks or `if-else` constructs.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```sol
433 if (end > length) {
434     end = length;
435 }
436
437 Unstake.UnstakeViewInfo[] memory result = new Unstake.UnstakeViewInfo[](end - start);
438
439 for (uint256 index = start; index < end; index++) {
440     result[index - start].index = unstakeList[account][index].index;
441     result[index - start].unstakerAddress = unstakeList[account][index].unstakerAddress;
442     result[index - start].unstakeTime = unstakeList[account][index].unstakeTime;
```

Example (Cont.):

```sol
443     result[index - start].nativeTokenAmount = unstakeList[account]
[index].nativeTokenAmount;
444     result[index - start].reStakingTokenAmount = unstakeList[account]
[index].reStakingTokenAmount;
445     result[index - start].claimTime = unstakeList[account][index].claimTime;
446     result[index - start].state = unstakeList[account][index].state;
447     result[index - start].pairTokenAmount = pairTokenAmountList[account][index];
448 }
```

Recommendation:

Given that safe arithmetics are toggled on by default in `pragma` versions of `0.8.X`, we advise the linked statements to be wrapped in `unchecked` code blocks thereby optimizing their execution cost.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The Lair Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

RSM-03C: Inefficient Initialization Flow

Type	Severity	Location
Gas Optimization	Informational	ReStakingManager.sol:L136, L143-L149, L151, L153

Description:

The `ReStakingManager::initialize` function will assign the `DEFAULT_ADMIN_ROLE` to the caller temporarily so as to be able to invoke several access-controlled functions and will revoke the role from the caller afterwards.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```sol
136 _grantRole(DEFAULT_ADMIN_ROLE, _msgSender());
137
138 gcKlayAddress = address(0x999999999939Ba65AbB254339eEc0b2A0daC80E9);
139 stKlayAddress = address(0xF80F2b22932fCEC6189b9153aA18662b15CC9C00);
140 sKlayAddress = address(0xA323d7386b671E8799dcA3582D6658FdcDcD940A);
141 sKlayGCAddress = payable(0x06cD16588aE09DEEa859Dcfc8b67386Bca412433);
142
143 setReStakingTokenAddress(_reStakingTokenAddress);
144 setStakingNodeAddress(_stakingNodeAddress);
145 setStakingTokenAddress(_stakingTokenAddress);
```

Example (Cont.):

```sol
146 setTransferAddress(_transferAddress);
147 setTreasuryAddress(_treasuryAddress);
148 setStakingNodeControllerAddress(_stakingNodeControllerAddress);
149 setClaimIntervalTime(_claimIntervalTime);
150
151 approveToken(_stakingTokenAddress, 100000000000000000000000000);
152
153 _revokeRole(DEFAULT_ADMIN_ROLE, _msgSender());
```

Recommendation:

We advise the code of each privileged function to be internalized and invoked by each implementation as well as the `ReStakingManager::initialize` function itself, optimizing the contract's initialization cost significantly.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The Lair Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

RSM-04C: Inefficient Loop Limit Evaluation

Type	Severity	Location
Gas Optimization	Informational	ReStakingManager.sol:L290

Description:

The linked `for` loop evaluates its limit inefficiently on each iteration.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```sol
290 for (uint256 i = 0; i < approvedTokenArray.length; i++) {
```

Recommendation:

We advise the statements within the `for` loop limit to be relocated outside to a local variable declaration that is consequently utilized for the evaluation to significantly reduce the codebase's gas cost. We should note the same optimization is applicable for storage reads present in such limits as they are newly read on each iteration (i.e. `length` members of arrays in storage).

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The loop limit's evaluation was optimized as advised, caching the `length` member of the `approvedTokenArray` outside the `for` loop's evaluation.

RSM-05C: Inefficient Variable Mutabilities

Type	Severity	Location
Gas Optimization	Informational	ReStakingManager.sol:L138-L141

Description:

The referenced variables are assigned-to only once within the `ReStakingManager` contract and specifically during the contract's **`ReStakingManager::initialize`** function to value literals.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```sol
122 function initialize(address _owner
123 , address _reStakingTokenAddress
124 , address _stakingNodeAddress
125 , address _stakingTokenAddress
126 , address payable _treasuryAddress
127 , address payable _transferAddress
128 , address payable _stakingNodeControllerAddress
129 , uint256 _claimIntervalTime
130 ) public initializer {
131     __Ownable_init_unchained(_owner);
```

Example (Cont.):

```sol
    __AccessControl_init_unchained();
    __Pausable_init_unchained();
    __ReentrancyGuard_init_unchained();

    _grantRole(DEFAULT_ADMIN_ROLE, _msgSender());

    gcKlayAddress = address(0x999999999939Ba65AbB254339eEc0b2A0daC80E9);
    stKlayAddress = address(0xF80F2b22932fCEC6189b9153aA18662b15CC9C00);
    sKlayAddress = address(0xA323d7386b671E8799dcA3582D6658FdcDcD940A);
    sKlayGCAddress = payable(0x06cD16588aE09DEEa859Dcfc8b67386Bca412433);
```

Recommendation:

We advise the assignments to be relocated to each variable's declaration directly and each variable to be set as `constant`, optimizing the code's gas cost significantly.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The Lair Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

RSM-06C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	**ReStakingManager.sol**: • I-1: **L287**, **L301-L304** • I-2: **L313**, **L315**, **L316**, **L318**, **L319** • I-3: **L424**, **L440-L447** • I-4: **L533**, **L534**, **L576**, **L577**, **L579**, **L580** • I-5: **L650** • I-6: **L679-L681** • I-7: **L697-L702**, **L704**, **L705**, **L707**, **L710**, **L711**, **L716**, **L718**, **L720**

Description:

The linked statements perform key-based lookup operations on `mapping` declarations from storage multiple times for the same key redundantly.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```
286 function approveToken(address tokenAddress, uint256 allowedTotalAmount) public
onlyRole(DEFAULT_ADMIN_ROLE) validAddress(tokenAddress)
validMoreThanZero(allowedTotalAmount) {
287     require(approvedTokenList[tokenAddress].state !=
State.ApprovedTokenState.Approved, "ReStakingManager:: already approved token");
288
289     bool isExist = false;
290     for (uint256 i = 0; i < approvedTokenArray.length; i++) {
291         if (approvedTokenArray[i].tokenAddress == tokenAddress) {
292             isExist = true;
293             break;
294         }
295     }
```

Example (Cont.):

```sol

    if (!isExist) {
        approvedTokenList[tokenAddress] = Token.TokenInfo(approvedTokenArray.length, tokenAddress, State.ApprovedTokenState.Approved, allowedTotalAmount, 0, 0);
        approvedTokenArray.push(approvedTokenList[tokenAddress]);
    } else {
        approvedTokenList[tokenAddress].state = State.ApprovedTokenState.Approved;
        approvedTokenArray[approvedTokenList[tokenAddress].index].state = approvedTokenList[tokenAddress].state;
        approvedTokenList[tokenAddress].allowedTotalAmount = allowedTotalAmount;
        approvedTokenArray[approvedTokenList[tokenAddress].index].allowedTotalAmount = approvedTokenList[tokenAddress].allowedTotalAmount;
    }
}
```

Recommendation:

As the lookups internally perform an expensive `keccak256` operation, we advise the lookups to be cached wherever possible to a single local declaration that either holds the value of the `mapping` in case of primitive types or holds a `storage` pointer to the `struct` contained.

As the compiler's optimizations may take care of these caching operations automatically at-times, we advise the optimization to be selectively applied, tested, and then fully adopted to ensure that the proposed caching model indeed leads to a reduction in gas costs.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The Lair Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

RSM-07C: Redundant Application of Modifiers

Type	Severity	Location
Gas Optimization	Informational	ReStakingManager.sol:L530

Description:

The `Validator::validAddress` modifier for the `tokenAddress` as well as the `Validator::validMoreThanZero` modifier for the `amount` variable are applied twice during a **`ReStakingManager::stake`** invocation.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```sol
506 function stake(address tokenAddress, uint256 amount, uint256 _pairRate, uint256 slippage) public validAddress(tokenAddress) validMoreThanZero(amount) nonReentrant whenNotPaused returns (uint256, uint256) {
507     require(getReStakingTokenAddress() != address(0), "ReStakingManager:: reStakingTokenAddress zero address2");
508     require(approvedTokenList[tokenAddress].state == State.ApprovedTokenState.Approved, "ReStakingManager:: not approved token");
509     require(slippage <= 10000, "ReStakingManager:: slippage is over 10000");
510
511     updatePairRate();
512
513     return _stake(_msgSender(), tokenAddress, amount, _pairRate, slippage, true);
514 }
515
```

Example (Cont.):

```sol
function getBalance(address token, address sender) public view returns (uint256)
{
    return IERC20(token).balanceOf(sender);
}

/**
  * @dev stake by reStaking token
  * @param sender sender
  * @param tokenAddress token address
  * @param amount amount
  * @param _pairRate pair rate
  * @param slippage slippage
  * @param isApproveCheck is approve token check
  * @return staking token amount, pair token amount
  */
function _stake(address sender, address tokenAddress, uint256 amount, uint256
_pairRate, uint256 slippage, bool isApproveCheck) private validAddress(sender)
validAddress(tokenAddress) validMoreThanZero(amount) returns (uint256, uint256) {
```

Recommendation:

We advise the modifiers to be omitted from the internal `ReStakingManager::_stake` implementation, optimizing the code's gas cost.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The Lair Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

RSM-08C: Redundant Balance & Allowance Measurements

Type	Severity	Location
Gas Optimization	Informational	**ReStakingManager.sol**: • I-1: **L556-L557** • I-2: **L566** • I-3: **L568-L569** • I-4: **L583** • I-5: **L618** • I-6: **L636** • I-7: **L701**

Description:

The referenced balance and allowance measurements are redundant as the ensuing **EIP-20** transfer or burn operations would fail if they are not satisfied.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```sol
555 require(frontPairAmount <= pairAmount && pairAmount <= frontPairSlippage, "ReStakingManager:: slippage is over");
556 require(getBalance(pairTokenAddress, sender) >= pairAmount, "ReStakingManager:: pairToken balance is not enough");
557 require(IERC20(pairTokenAddress).allowance(sender, address(this)) >= pairAmount, "ReStakingManager:: pairToken allowance is not enough");
558
559 totalPairTokenStakedAmount = totalPairTokenStakedAmount.add(pairAmount);
560 pairTokenUserStakeAmount = pairTokenUserStakeAmount.add(pairAmount);
561
562 IERC20(pairTokenAddress).safeTransferFrom(sender, getTreasuryAddress(), pairAmount);
```

Recommendation:

We advise them to be omitted, optimizing the code's gas cost.

Alleviation (09c58d1f17):

The Lair Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

Alleviation (803a3419a9):

Several of the redundant balance measurements have been omitted whilst the redundant allowance evaluations remain, rendering this exhibit to be partially addressed and thus acknowledged.

RSM-09C: Redundant Storage Reads

Type	Severity	Location
Gas Optimization	Informational	**ReStakingManager.sol**: • I-1: **L302**, **L304** • I-2: **L316**, **L319** • I-3: **L331**

Description:

The referenced storage read statements are inefficient as the value those storage variables hold are already known and present in memory.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```
315 approvedTokenList[tokenAddress].state = State.ApprovedTokenState.Removed;
316 approvedTokenArray[approvedTokenList[tokenAddress].index].state =
approvedTokenList[tokenAddress].state;
317
318 approvedTokenList[tokenAddress].allowedTotalAmount = 0;
319 approvedTokenArray[approvedTokenList[tokenAddress].index].allowedTotalAmount =
approvedTokenList[tokenAddress].allowedTotalAmount;
```

Recommendation:

We advise the storage reads to be replaced by the values themselves, optimizing each function's gas cost.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The Lair Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

RSM-10C: Redundant Zero-Value Assignments

Type	Severity	Location
Gas Optimization	Informational	ReStakingManager.sol:L158-L164

Description:

The referenced zero-value assignments are redundant as the relevant variables already contain `0` within them.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```sol
158 totalStakedAmount = 0;
159 totalUnstakedAmount = 0;
160 totalClaimedAmount = 0;
161 totalStakeCount = 0;
162 totalUnstakeCount = 0;
163 totalClaimCount = 0;
164 bufferTokenAmount = 0;
```

Recommendation:

We advise the code to remove these assignments as long as these variables are not meant to be cleared as part of an upgrade.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The Lair Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

RSM-11C: Repetitive Value Literals

Type	Severity	Location
Code Style	Informational	ReStakingManager.sol:L65, L509, L549, L779, L780

Description:

The linked value literals are repeated across the codebase multiple times.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```sol
65  uint256 private constant DECIMALS = 10**18;
```

Recommendation:

We advise each to be set to its dedicated `constant` variable instead, optimizing the legibility of the codebase.

In case some of the `constant` declarations have already been introduced, we advise them to be properly re-used across the code.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The Lair Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

RSM-12C: Sub-Optimal Early Return

Type	Severity	Location
Gas Optimization	Informational	ReStakingManager.sol:L426, L433-L435

Description:

The early return statement of the `ReStakingManager::getUnstakeInfos` function is meant to optimize its gas cost yet fails to do so adequately. Specifically, the case whereby `start == end` is not covered and results in the redundant execution of several statements.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```sol
424 uint256 length = unstakeList[account].length;
425
426 if (length == 0 || count == 0 || page * count > length) {
427     return new Unstake.UnstakeViewInfo[](0);
428 }
429
430 uint256 start = page * count;
431 uint256 end = start + count;
432
433 if (end > length) {
```

Example (Cont.):

```sol
434     end = length;
435 }
436
437 Unstake.UnstakeViewInfo[] memory result = new Unstake.UnstakeViewInfo[](end - start);
438
439 for (uint256 index = start; index < end; index++) {
440     result[index - start].index = unstakeList[account][index].index;
441     result[index - start].unstakerAddress = unstakeList[account][index].unstakerAddress;
442     result[index - start].unstakeTime = unstakeList[account][index].unstakeTime;
443     result[index - start].nativeTokenAmount = unstakeList[account][index].nativeTokenAmount;
444     result[index - start].reStakingTokenAmount = unstakeList[account][index].reStakingTokenAmount;
445     result[index - start].claimTime = unstakeList[account][index].claimTime;
446     result[index - start].state = unstakeList[account][index].state;
447     result[index - start].pairTokenAmount = pairTokenAmountList[account][index];
448 }
449
450 return result;
```

Recommendation:

We advise the conditional to be relocated after the `start` and `end` calculations, and the code to simply check if `start >= end` in which case the code should return an empty array.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The Lair Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

RSM-13C: Suboptimal Struct Declaration Styles

Type	Severity	Location
Code Style	Informational	**ReStakingManager.sol**: • I-1: **L298** • I-2: **L650**

Description:

The linked declaration styles of the referenced structs are using index-based argument initialization.

Example:

contracts/restaking/ReStakingManager/ReStakingManager.sol

SOL

```
298 approvedTokenList[tokenAddress] = Token.TokenInfo(approvedTokenArray.length,
tokenAddress, State.ApprovedTokenState.Approved, allowedTotalAmount, 0, 0);
```

Recommendation:

We advise the key-value declaration format to be utilized instead in each instance, greatly increasing the legibility of the codebase.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The Lair Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

ReStakingTreasury Code Style Findings

RST-01C: Deprecated Revert Pattern

Type	Severity	Location
Code Style	Informational	**ReStakingTreasury.sol**: • I-1: **L147** • I-2: **L155** • I-3: **L164** • I-4: **L172**

Description:

The referenced statements will issue a `revert` with a textual message rather than an `error` which has been deprecated.

Example:

contracts/restaking/Treasury/ReStakingTreasury.sol

SOL

```
147 revert("ReStakingTreasury:: ownership cannot be renounced");
```

Recommendation:

We advise a proper `error` to be declared for each instance and to be emitted, optimizing the code's syntax.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The Lair Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

RST-02C: Inefficient Balance Evaluation

Type	Severity	Location
Gas Optimization	Informational	ReStakingTreasury.sol:L99

Description:

The referenced balance evaluation is inefficient as a successful **EIP-20** transfer guarantees adequate balance was present in the contract.

Example:

contracts/restaking/Treasury/ReStakingTreasury.sol

SOL

```sol
99  require(IERC20(tokenAddress).balanceOf(address(this)) >= amount,
"ReStakingTransfer:: insufficient balance");
100
101 IERC20(tokenAddress).safeTransfer(receiver, amount);
```

Recommendation:

We advise it to be omitted, optimizing the code's gas cost.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The Lair Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

RST-03C: Inefficient Initialization Flow

Type	Severity	Location
Gas Optimization	Informational	ReStakingTreasury.sol:L51, L53-L54, L56

Description:

The `ReStakingTreasury::initialize` function is inefficient as it will set the caller as the `DEFAULT_ADMIN_ROLE` temporarily so as to configure the relevant manager and token addresses.

Example:

contracts/restaking/Treasury/ReStakingTreasury.sol

SOL

```sol
51  _grantRole(DEFAULT_ADMIN_ROLE, _msgSender());
52
53  setReStakingManagerAddress(_reStakingManagerAddress);
54  setStakingTokenAddress(_stakingTokenAddress);
55
56  _revokeRole(DEFAULT_ADMIN_ROLE, _msgSender());
```

Recommendation:

We advise the code of each function to be internalized and the `ReStakingTreasury::initialize` function to invoke those internal variants, optimizing the code significantly.

Alleviation (09c58d1f17cf988eb3a99a4caf7567cafc2e44c7):

The Lair Finance team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

Finding Types

A description of each finding type included in the report can be found below and is linked by each respective finding. A full list of finding types Omniscia has defined will be viewable at the central audit methodology we will publish soon.

Input Sanitization

As there are no inherent guarantees to the inputs a function accepts, a set of guards should always be in place to sanitize the values passed in to a particular function.

Indeterminate Code

These types of issues arise when a linked code segment may not behave as expected, either due to mistyped code, convoluted if blocks, overlapping functions / variable names and other ambiguous statements.

Language Specific

Language specific issues arise from certain peculiarities that the Circom language boasts that discerns it from other conventional programming languages.

Curve Specific

Circom defaults to using the BN128 scalar field (a 254-bit prime field), but it also supports BSL12-381 (which has a 255-bit scalar field) and Goldilocks (with a 64-bit scalar field). However, since there are no constants denoting either the prime or the prime size in bits available in the Circom language, some Circomlib templates like `Sign` (which returns the sign of the input signal), and `AliasCheck` (used by the strict versions of `Num2Bits` and `Bits2Num`), hardcode either the BN128 prime size or some other constant related to BN128. Using these circuits with a custom prime may thus lead to unexpected results and should be avoided.

Code Style

In these types of findings, we identify whether a project conforms to a particular naming convention and whether that convention is consistent within the codebase and legible. In case of inconsistencies, we point them out under this category. Additionally, variable shadowing falls under this category as well which is identified when a local-level variable contains the same name as a toplevel variable in the circuit.

Mathematical Operations

This category is used when a mathematical issue is identified. This implies an issue with the implementation of a calculation compared to the specifications.

Logical Fault

This category is a bit broad and is meant to cover implementations that contain flaws in the way they are implemented, either due to unimplemented functionality, unaccounted-for edge cases or similar extraordinary scenarios.

Privacy Concern

This category is used when information that is meant to be kept private is made public in some way.

Proof Concern

Under-constrained signals are one of the most common issues in zero-knowledge circuits. Issues with proof generation fall under this category.

Severity Definition

In the ever-evolving world of blockchain technology, vulnerabilities continue to take on new forms and arise as more innovative projects manifest, new blockchain-level features are introduced, and novel layer-2 solutions are launched. When performing security reviews, we are tasked with classifying the various types of vulnerabilities we identify into subcategories to better aid our readers in understanding their impact.

Within this page, we will clarify what each severity level stands for and our approach in categorizing the findings we pinpoint in our audits. To note, all severity assessments are performed **as if the contract's logic cannot be upgraded** regardless of the underlying implementation.

Severity Levels

There are five distinct severity levels within our reports; `unknown`, `informational`, `minor`, `medium`, and `major`. A TL;DR overview table can be found below as well as a dedicated chapter to each severity level:

	Impact (None)	Impact (Low)	Impact (Moderate)	Impact (High)
Likelihood (None)	Informational	Informational	Informational	Informational
Likelihood (Low)	Informational	Minor	Minor	Medium
Likelihood (Moderate)	Informational	Minor	Medium	Major
Likelihood (High)	Informational	Medium	Major	Major

Unknown Severity

The `unknown` severity level is reserved for misbehaviors we observe in the codebase that cannot be quantified using the above metrics. Examples of such vulnerabilities include potentially desirable system behavior that is undocumented, reliance on external dependencies that are out-of-scope but could result in some form of vulnerability arising, use of external out-of-scope contracts that appears incorrect but cannot be pinpointed, and other such vulnerabilities.

In general, `unknown` severity level vulnerabilities require follow-up information by the project being audited and are either adjusted in severity (if valid), or marked as nullified (if invalid).

Additionally, the `unknown` severity level is sometimes assigned to centralization issues that cannot be assessed in likelihood due to their exploitation being tied to the honesty of the project's team.

Informational Severity

The `informational` severity level is dedicated to findings that do not affect the code functionally and tend to be stylistic or optimizational in nature. Certain edge cases are also set under `informational` vulnerabilities, such as overflow operations that will not manifest in the lifetime of the contract but should be guarded against as a best practice, to give an example.

Minor Severity

The `minor` severity level is meant for vulnerabilities that require functional changes in the code but tend to either have little impact or be unlikely to be recreated in a production environment. These findings can be acknowledged except for findings with a moderate impact but low likelihood which must be alleviated.

Medium Severity

The `medium` severity level is assigned to vulnerabilities that must be alleviated and have an observable impact on the overall project. These findings can only be acknowdged if the project deems them desirable behavior and we disagree with their point-of-view, instead urging them to reconsider their stance while marking the exhibit as acknowledged given that the project has ultimate say as to what vulnerabilities they end up patching in their system.

Major Severity

The `major` severity level is the maximum that can be specified for a finding and indicates a significant flaw in the code that must be alleviated.

Likelihood & Impact Assessment

As the preface chapter specifies, the blockchain space is constantly reinventing itself meaning that new vulnerabilities take place and our understanding of what security means differs year-to-year.

In order to reliably assess the likelihood and impact of a particular vulnerability, we instead apply an abstract measurement of a vulnerability's impact, duration the impact is applied for, and probability that the vulnerability would be exploited in a production environment.

Our proposed definitions are inspired by multiple sources in the security community and are as follows:

- Impact (High): A core invariant of the protocol can be broken for an extended duration.
- Impact (Moderate): A non-core invariant of the protocol can be broken for an extended duration or at scale, or an otherwise major-severity issue is reduced due to hypotheticals or external factors affecting likelihood.
- Impact (Low): A non-core invariant of the protocol can be broken with reduced likelihood or impact.
- Impact (None): A code or documentation flaw whose impact does not achieve low severity, or an issue without theoretical impact; a valuable best-practice
- Likelihood (High): A flaw in the code that can be exploited trivially and is ever-present.
- Likelihood (Moderate): A flaw in the code that requires some external factors to be exploited that are likely to manifest in practice.
- Likelihood (Low): A flaw in the code that requires multiple external factors to be exploited that may manifest in practice but would be unlikely to do so.
- Likelihood (None): A flaw in the code that requires external factors proven to be impossible in a production environment, either due to mathematical constraints, operational constraints, or system-related factors (i.e. EIP-20 tokens not being re-entrant).

Disclaimer

The following disclaimer applies to all versions of the audit report produced (preliminary / public / private) and is in effect for all past, current, and future audit reports that are produced and hosted under Omniscia:

IMPORTANT TERMS & CONDITIONS REGARDING OUR SECURITY AUDITS/REVIEWS/REPORTS AND ALL PUBLIC/PRIVATE CONTENT/DELIVERABLES

Omniscia ("Omniscia") has conducted an independent security review to verify the integrity of and highlight any vulnerabilities, bugs or errors, intentional or unintentional, that may be present in the codebase that were provided for the scope of this Engagement.

Blockchain technology and the cryptographic assets it supports are nascent technologies. This makes them extremely volatile assets. Any assessment report obtained on such volatile and nascent assets may include unpredictable results which may lead to positive or negative outcomes.

In some cases, services provided may be reliant on a variety of third parties. This security review does not constitute endorsement, agreement or acceptance for the Project and technology that was reviewed. Users relying on this security review should not consider this as having any merit for financial advice or technological due diligence in any shape, form or nature.

The veracity and accuracy of the findings presented in this report relate solely to the proficiency, competence, aptitude and discretion of our auditors. Omniscia and its employees make no guarantees, nor assurance that the contracts are free of exploits, bugs, vulnerabilities, deprecation of technologies or any system / economical / mathematical malfunction.

This audit report shall not be printed, saved, disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Omniscia.

All the information/opinions/suggestions provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report.

Information in this report is provided 'as is'. Omniscia is under no covenant to the completeness, accuracy or solidity of the contracts reviewed. Omniscia's goal is to help reduce the attack vectors/surface and the high level of variance associated with utilizing new and consistently changing technologies.

Omniscia in no way claims any guarantee, warranty or assurance of security or functionality of the technology that was in scope for this security review.

In no event will Omniscia, its partners, employees, agents or any parties related to the design/creation of this security review be ever liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this security review.

Cryptocurrencies and all other technologies directly or indirectly related to cryptocurrencies are not standardized, highly prone to malfunction and extremely speculative by nature. No due diligence and/or safeguards may be insufficient and users should exercise maximum caution when participating and/or investing in this nascent industry.

The preparation of this security review has made all reasonable attempts to provide clear and actionable recommendations to the Project team (the “client”) with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts in scope for this engagement.

It is the sole responsibility of the Project team to provide adequate levels of test and perform the necessary checks to ensure that the contracts are functioning as intended, and more specifically to ensure that the functions contained within the contracts in scope have the desired intended effects, functionalities and outcomes, as documented by the Project team.

All services, the security reports, discussions, work product, attack vectors description or any other materials, products or results of this security review engagement is provided "as is" and "as available" and with all faults, uncertainty and defects without warranty or guarantee of any kind.

Omniscia will assume no liability or responsibility for delays, errors, mistakes, or any inaccuracies of content, suggestions, materials or for any loss, delay, damage of any kind which arose as a result of this engagement/security review.

Omniscia will assume no liability or responsibility for any personal injury, property damage, of any kind whatsoever that resulted in this engagement and the customer having access to or use of the products, engineers, services, security report, or any other other materials.

For avoidance of doubt, this report, its content, access, and/or usage thereof, including any associated services or materials, shall not be considered or relied upon as any form of financial, investment, tax, legal, regulatory, or any other type of advice.