OMNISCIA

March 11, 2025

SMART CONTRACT AUDIT REPORT

Stake DAO

ZeroLend Integration


omniscia.io


info@omniscia.io


Online report: stake-dao-zerolend-integration

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: stake-dao-zerolend-integration

ZeroLend Integration Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
5bfeba2843	February 21st 2025	749619c792
afe64e1236	March 6th 2025	f980d9ee8e
e4df6cf928	March 11th 2025	b05a4939f6

Audit Overview

We were tasked with performing an audit of the Stake DAO codebase and in particular their ZeroLend Integration module.

The `Depositor` and `Accumulator` implementations are expected to integrate with the ZeroLend `ZEROvp` infrastructure, including the `T-ZERO` NFT as well as base `ZERO` token.

The `Depositor` contract will continuously create and merge all NFT lock positions within the ZeroLend system that arise from its usage, maintaining a single NFT ID at all times that amalgamates all deposits of the contract.

The `Accumulator` contract will permit rewards to be claimed via it from the `ZEROvp` system, charge fees on them, and properly distribute them via the relevant gauge implementation.

For the purposes of the engagement, we considered the out-of-scope gauge implementation to function per its usage across the codebase and we did not validate its implementation.

Over the course of the audit, we identified a significant issue albeit with minimal impact in the way multiple input NFT IDs are merged into a single one within the `Depositor` contract and specifically how the NFT's final lock end time is measured.

We advise the Stake DAO 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 Stake DAO 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 Stake DAO and have identified that a single exhibit has been partially addressed. We advise the Stake DAO team to revisit the following exhibit: `ARO-02M`

Post-Audit Conclusion (e4df6cf928)

The Stake DAO team provided us with a follow-up commit hash to evaluate the alleviation of `ARO-02M` on.

We consider the alleviation to have been applied properly, and all outputs of the audit report to have been properly consumed by the Stake DAO team.

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	0	0	0	0
Informational	16	13	0	3
Minor	5	3	0	2
Medium	0	0	0	0
Major	0	0	0	0

During the audit, we filtered and validated a total of **8 findings utilizing static analysis** tools as well as identified a total of **13 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/stake-dao/contracts-monorepo**
- Commit: 5bfeba2843ea82082cf6978dbc3a725d17c62208
- Language: Solidity
- Network: Ethereum, Linea
- Revisions: **5bfeba2843**, **afe64e1236**, **e4df6cf928**

Contracts Assessed

File	Total Finding(s)
packages/lockers/src/linea/zerolend/Accumulator.sol (ARO)	3
packages/lockers/src/common/depositor/BaseDepositor.sol (BDR)	6
packages/lockers/src/common/accumulator/BaseAccumulator.sol (BAR)	9
packages/lockers/src/linea/zerolend/Depositor.sol (DRO)	3

Compilation

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

To compile the project, the `build` command needs to be issued via the `forge` CLI tool:

```bash
forge build
```

The `forge` tool automatically selects Solidity version `0.8.19` based on the version specified within the `foundry.toml` file.

The project contains discrepancies with regards to the Solidity version used as the `pragma` statements of the contracts are open-ended (`^0.8.19`).

We advise them to be locked to `0.8.19` (`=0.8.19`), the same version utilized for our static analysis as well as optimizational review of the codebase.

During compilation with the `foundry` 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 **41 potential issues** within the codebase of which **18 were ruled out to be false positives** or negligible findings.

The remaining **23 issues** were validated and grouped and formalized into the **8 exhibits** that follow:

ID	Severity	Addressed	Title
BAR-01S	Informational	Yes	Illegible Numeric Value Representation
BAR-02S	Informational	Yes	Inexistent Event Emissions
BAR-03S	Informational	Yes	Inexistent Sanitization of Input Addresses
BAR-04S	Informational	Yes	Inexistent Visibility Specifier
BDR-01S	Informational	Yes	Illegible Numeric Value Representation
BDR-02S	Informational	Yes	Inexistent Event Emissions
BDR-03S	Informational	Yes	Inexistent Sanitization of Input Addresses
DRO-01S	Informational	Yes	Inexistent Sanitization of Input Addresses

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Stake DAO's ZeroLend integration.

As the project at hand implements modules that integrate with the ZeroLend 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 as well as ZeroLend'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 **did not identify any significant vulnerabilities** within the system beyond a flaw in lock end time calculations that does not arise in any non-negligible impact.

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.

A total of **13 findings** were identified over the course of the manual review of which **6 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
ARO-01M	Minor	Acknowledged	Inexistent Validation of Input Arguments
ARO-02M	Minor	Yes	Inexistent Validation of Module Transaction Success
BAR-01M	Minor	Yes	Inexistent Validation of Fee Configurations
BAR-02M	Minor	Acknowledged	Inflexible Claimer Fee
BDR-01M	Informational	Yes	Non-Standard Input Validation

DRO-01M	Minor	Yes	Incorrect Evaluation of Lock End

Code Style

During the manual portion of the audit, we identified **7 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
ARO-01C	Informational	Acknowledged	Redundant Self-Transfers
BAR-01C	Informational	Acknowledged	Empty Implementation of Virtual Function
BAR-02C	Informational	Yes	Inefficient Data Structure
BAR-03C	Informational	Yes	Suboptimal Struct Declaration Style
BDR-01C	Informational	Yes	Generic Typographic Mistake
BDR-02C	Informational	Acknowledged	Potential Increase of Capability
DRO-01C	Informational	Yes	Unused Error Declaration

BaseAccumulator Static Analysis Findings

BAR-01S: Illegible Numeric Value Representation

Type	Severity	Location
Code Style	Informational	BaseAccumulator.sol:L109

Description:

The linked representation of a numeric literal is sub-optimally represented decreasing the legibility of the codebase.

Example:

packages/lockers/src/common/accumulator/BaseAccumulator.sol

```sol
109 claimerFee = 1e15; // 0.1%
```

Recommendation:

To properly illustrate the 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 (afe64e12363024e853cf5e0174506e03182346a3):

The referenced value literal has been updated in its representation to `0.001e18` in accordance with the recommendation's underscore style, addressing this exhibit.

BAR-02S: Inexistent Event Emissions

Type	Severity	Location
Language Specific	Informational	**BaseAccumulator.sol**: • I-1: **L208-L210** • I-2: **L214-L216** • I-3: **L221-L224** • I-4: **L228-L232** • I-5: **L243-L246** • I-6: **L248-L250**

Description:

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

Example:

packages/lockers/src/common/accumulator/BaseAccumulator.sol

SOL

```sol
208 function setDistributor(address _distributor) external onlyGovernance {
209     sdtDistributor = _distributor;
210 }
```

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 (afe64e12363024e853cf5e0174506e03182346a3):

The `ClaimerFeeUpdated`, `SDTDistributorUpdated`, `FeeReceiverUpdated`, `GovernanceUpdateProposed`, `GovernanceUpdateAccepted`, `RewardTokenApproved`, `FeeSplitUpdated`, and `StrategyUpdated` events were introduced to the codebase and are correspondingly emitted in the **`BaseAccumulator::setClaimerFee`**, **`BaseAccumulator::setDistributor`**, **`BaseAccumulator::setFeeReceiver`**, **`BaseAccumulator::transferGovernance`**, **`BaseAccumulator::acceptGovernance`**, **`BaseAccumulator::setFeeSplit`**, and **`BaseAccumulator::setStrategy`** functions respectively, addressing this exhibit in full.

BAR-03S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	BaseAccumulator.sol:L102-L110

Description:

The linked function(s) accept `address` arguments yet do not properly sanitize them.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

packages/lockers/src/common/accumulator/BaseAccumulator.sol

SOL

```
102 constructor(address _gauge, address _rewardToken, address _locker, address
_governance) {
103     gauge = _gauge;
104     locker = _locker;
105     rewardToken = _rewardToken;
106
107     governance = _governance;
108
109     claimerFee = 1e15; // 0.1%
110 }
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that each `address` specified is non-zero.

Alleviation (afe64e12363024e853cf5e0174506e03182346a3):

All input arguments of the `BaseAccumulator::constructor` function are adequately sanitized as non-zero in the latest in-scope revision of the codebase, addressing this exhibit.

BAR-04S: Inexistent Visibility Specifier

Type	Severity	Location
Code Style	Informational	BaseAccumulator.sol:L30

Description:

The linked variable has no visibility specifier explicitly set.

Example:

packages/lockers/src/common/accumulator/BaseAccumulator.sol

SOL

```
30  Split feeSplit;
```

Recommendation:

We advise one to be set so to avoid potential compilation discrepancies in the future as the current behaviour is for the compiler to assign one automatically which may deviate between `pragma` versions.

Alleviation (afe64e12363024e853cf5e0174506e03182346a3):

The `private` visibility specifier has been introduced to the referenced variable, preventing potential compilation discrepancies and addressing this exhibit.

BaseDepositor Static Analysis Findings

BDR-01S: Illegible Numeric Value Representation

Type	Severity	Location
Code Style	Informational	BaseDepositor.sol:L19

Description:

The linked representation of a numeric literal is sub-optimally represented decreasing the legibility of the codebase.

Example:

packages/lockers/src/common/depositor/BaseDepositor.sol

```sol
19  uint256 public constant DENOMINATOR = 10_000;
```

Recommendation:

To properly illustrate the 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 (afe64e12363024e853cf5e0174506e03182346a3):

The referenced value literal has been updated in its representation to `1e18` in accordance with the recommendation's underscore style, addressing this exhibit.

BDR-02S: Inexistent Event Emissions

Type	Severity	Location
Language Specific	Informational	**BaseDepositor.sol**: • I-1: **L203-L205** • I-2: **L208-L214** • I-3: **L224-L230** • I-4: **L234-L238**

Description:

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

Example:

packages/lockers/src/common/depositor/BaseDepositor.sol

SOL

```sol
203 function transferGovernance(address _governance) external onlyGovernance {
204     futureGovernance = _governance;
205 }
```

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 (afe64e12363024e853cf5e0174506e03182346a3):

The `GovernanceUpdateProposed`, `GovernanceUpdateAccepted`, `GaugeUpdated`, and `LockIncentiveUpdated` events were introduced to the codebase and are correspondingly emitted in the **`BaseDepositor::transferGovernance`**, **`BaseDepositor::acceptGovernance`**, **`BaseDepositor::setGauge`**, and **`BaseDepositor::setFees`** functions respectively, addressing this exhibit in full.

BDR-03S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	BaseDepositor.sol:L70-L84

Description:

The linked function(s) accept `address` arguments yet do not properly sanitize them.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

packages/lockers/src/common/depositor/BaseDepositor.sol

SOL

```sol
70  constructor(address _token, address _locker, address _minter, address _gauge,
uint256 _maxLockDuration) {
71      governance = msg.sender;
72
73      token = _token;
74      gauge = _gauge;
75      minter = _minter;
76      locker = _locker;
77
78      MAX_LOCK_DURATION = _maxLockDuration;
79
```

Example (Cont.):

```sol
80      /// Approve sdToken to gauge.
81      if (gauge != address(0)) {
82          SafeTransferLib.safeApprove(minter, gauge, type(uint256).max);
83      }
84  }
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that each `address` specified is non-zero.

Alleviation (afe64e12363024e853cf5e0174506e03182346a3):

All input arguments of the `BaseDepositor::constructor` function are adequately sanitized as non-zero in the latest in-scope revision of the codebase, addressing this exhibit.

Depositor Static Analysis Findings

DRO-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	**Depositor.sol:L62-L67**

Description:

The linked function(s) accept `address` arguments yet do not properly sanitize them.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

packages/lockers/src/linea/zerolend/Depositor.sol

SOL

```sol
62  constructor(address _token, address _locker, address _minter, address _gauge,
address _zeroLocker, address _veToken)
63      BaseDepositor(_token, _locker, _minter, _gauge, 4 * 365 days)
64  {
65      zeroLocker = ILockerToken(_zeroLocker);
66      veToken = IZeroVp(_veToken);
67  }
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that each `address` specified is non-zero.

Alleviation (afe64e12363024e853cf5e0174506e03182346a3):

All input arguments of the `Depositor::constructor` function are adequately sanitized as non-zero in the latest in-scope revision of the codebase, addressing this exhibit.

Accumulator Manual Review Findings

ARO-01M: Inexistent Validation of Input Arguments

Type	Severity	Location
Logical Fault	Minor	Accumulator.sol:L47, L69, L80

Description:

The `Accumulator::claimAndNotifyAll` function implementation will not validate that its input arguments match the `false` input arguments transmitted to the `BaseAccumulator::notifyReward` calls.

Impact:

Any invocation of the `Accumulator::claimAndNotifyAll` function with a `true` argument will result in success even though no notification of the SDT or claim of the fee strategy occurred.

Example:

packages/lockers/src/linea/zerolend/Accumulator.sol

SOL

```sol
47  function claimAndNotifyAll(bool, bool) external override {
48      uint256 lockerZeroBalanceBefore = IERC20(ZERO).balanceOf(locker);
49      uint256 lockerWethBalanceBefore = IERC20(WETH).balanceOf(locker);
50
51      // Claim rewards from the locker.
52      ILocker(locker).execTransactionFromModuleReturnData(
53          ZERO_VP, 0, abi.encodeWithSelector(IZeroVp.getReward.selector),
Enum.Operation.Call
54      );
55
56      // Get rewards amount.
```

Example (Cont.):

```sol
57      uint256 zeroReward = IERC20(ZERO).balanceOf(locker) -
lockerZeroBalanceBefore;
58      uint256 wethReward = IERC20(WETH).balanceOf(locker) -
lockerWethBalanceBefore;
59
60      // Transfer rewards to the accumulator.
61      if (zeroReward > 0) {
62          ILocker(locker).execTransactionFromModuleReturnData(
63              ZERO,
64              0,
65              abi.encodeWithSelector(IERC20.transfer.selector, address(this),
zeroReward),
66              Enum.Operation.Call
67          );
68
69          notifyReward(ZERO, false, false);
70      }
71
72      if (wethReward > 0) {
73          ILocker(locker).execTransactionFromModuleReturnData(
74              WETH,
75              0,
76              abi.encodeWithSelector(IERC20.transfer.selector, address(this),
wethReward),
77              Enum.Operation.Call
78          );
79
80          notifyReward(WETH, false, false);
81      }
82  }
```

Recommendation:

We advise such validation to be introduced as the function can presently be invoked with `true` arguments even though they are not utilized, misleading callers by the call's success.

Alleviation (afe64e12363024e853cf5e0174506e03182346a3):

The Stake DAO team evaluated this exhibit and its ramifications and opted to acknowledge it.

ARO-02M: Inexistent Validation of Module Transaction Success

Type	Severity	Location
Logical Fault	Minor	**Accumulator.sol**: • I-1: **L52** • I-2: **L62** • I-3: **L73**

Description:

The `Accumulator::claimAndNotifyAll` function does not validate that the Safe module interactions have been successfully executed, continuing execution as if they always succeed.

Impact:

Safe module interactions are not validated as having been successfully executed, causing the code to execute improperly as a no-op if they failed.

Example:

packages/lockers/src/linea/zerolend/Accumulator.sol

SOL

```sol
51  // Claim rewards from the locker.
52  ILocker(locker).execTransactionFromModuleReturnData(
53      ZERO_VP, 0, abi.encodeWithSelector(IZeroVp.getReward.selector),
Enum.Operation.Call
54  );
```

Recommendation:

We advise proper validation of the yielded `bool` to be performed, ensuring that the function executes as expected.

Alleviation (afe64e1236):

The code was updated to address solely the first of the three highlighted instances, rendering this exhibit partially addressed.

Alleviation (e4df6cf928):

The Stake DAO team proceeded with introducing validation of the `bool` variable yielded from the `ILocker::execTransactionFromModuleReturnData` function for the two remaining calls, alleviating this exhibit in full.

BaseAccumulator Manual Review Findings

BAR-01M: Inexistent Validation of Fee Configurations

Type	Severity	Location
Input Sanitization	Minor	BaseAccumulator.sol:L203, L245

Description:

The fee configurations of the `BaseAccumulator` are not validated to sum at most to the `DENOMINATOR`, permitting fees that cannot be charged to be defined.

Impact:

The `BaseAccumulator` can be configured in a state that would result in failure whenever fees are attempted to be applied.

Example:

packages/lockers/src/common/accumulator/BaseAccumulator.sol

SOL

```
240 /// @notice Set fee split
241 /// @param receivers array of receivers
242 /// @param fees array of fees
243 function setFeeSplit(address[] calldata receivers, uint256[] calldata fees)
external onlyGovernance {
244     if (receivers.length == 0 || receivers.length != fees.length) revert
INVALID_SPLIT();
245     feeSplit = Split(receivers, fees);
246 }
```

Recommendation:

We advise fees to be validated by ensuring that the sum of all fees charged do not exceed the `DENOMINATOR` whenever they are configured.

Alleviation (afe64e12363024e853cf5e0174506e03182346a3):

The code of the **`BaseAccumulator::setFeeSplit`** function was updated to sum the fees configured for each recipient and ensure that they total at most the `DENOMINATOR`, however, this partially addresses the issue.

As the first link of the exhibit highlights, the `claimerFee` must be taken into account within the sum as well per the implementation of **`BaseAccumulator::_chargeFee`**, rendering the current validation insufficient.

To note, the configuration of the `claimerFee` is meant to be sanitized as well in a similar regard, although we consider the individual fee split configurations to be more prone to errors.

BAR-02M: Inflexible Claimer Fee

Type	Severity	Location
Logical Fault	Minor	BaseAccumulator.sol:L128, L130, L145, L148, L180, L181

Description:

The claimer fee charged by the `BaseAccumulator::_chargeFee` function does not take into account whether the caller notified the `sdtDistributor` nor whether they claimed from the fee strategy.

Impact:

A user who invokes `BaseAccumulator::notifyReward` has no incentive to set `notifySDT` and `claimFeeStrategy` as enabled because their reward remains constant.

Example:

packages/lockers/src/common/accumulator/BaseAccumulator.sol

SOL

```sol
123 function notifyReward(address token, bool notifySDT, bool claimFeeStrategy)
public virtual {
124     uint256 amount = ERC20(token).balanceOf(address(this));
125     // notify token as reward in sdToken gauge
126     _notifyReward(token, amount, claimFeeStrategy);
127
128     if (notifySDT) {
129         // notify SDT
130         _distributeSDT();
131     }
132 }
```

Recommendation:

We advise the code to distribute a different claim reward depending on the gas expenditure of the caller (i.e. whether they triggered a full claim or a claim without notifying all contracts), ensuring users are incentivized to claim and notify all system components.

Alleviation (afe64e12363024e853cf5e0174506e03182346a3):

The Stake DAO team evaluated this exhibit and its ramifications and opted to acknowledge it.

BaseDepositor Manual Review Findings

BDR-01M: Non-Standard Input Validation

Type	Severity	Location
Standard Conformity	Informational	BaseDepositor.sol:L235

Description:

The referenced conditional will permit a fee adjustment to occur solely when it is valid, causing the transaction to succeed as a no-op otherwise.

Example:

packages/lockers/src/common/depositor/BaseDepositor.sol

SOL

```sol
234 function setFees(uint256 _lockIncentive) external onlyGovernance {
235     if (_lockIncentive >= 0 && _lockIncentive <= 30) {
236         lockIncentivePercent = _lockIncentive;
237     }
238 }
```

Recommendation:

We advise the transaction to fail if an invalid `_lockIncentive` have been set, ensuring callers do not mistakenly assume that their newly defined lock incentive has been set even though the transaction did not update it.

Alleviation (afe64e12363024e853cf5e0174506e03182346a3):

The validation pattern was updated to an `if-revert` check, addressing this exhibit.

Depositor Manual Review Findings

DRO-01M: Incorrect Evaluation of Lock End

Type	Severity	Location
Logical Fault	Minor	Depositor.sol:L182-L183

Description:

The `Depositor::_mergeLocksAndStake` function will improperly evaluate the `_currentLockEnd` of the stake we are merging from after it has been merged, resulting in a value of `0` being read consistently.

Impact:

As the updated `_lockEnd` is solely utilized for the emission of an event, the severity of this exhibit is capped at minor even though it illustrates a potentially significant error.

Example:

packages/lockers/src/linea/zerolend/Depositor.sol

SOL

```sol
176 for (uint256 index = 0; index < _tokenIds.length;) {
177     if (zeroLocker.ownerOf(_tokenIds[index]) != msg.sender) revert
NotOwnerOfToken(_tokenIds[index]);
178
179     _amount += zeroLocker.locked(_tokenIds[index]).amount;
180     _merge(_tokenIds[index], zeroLockedTokenId);
181
182     uint256 _currentLockEnd = zeroLocker.lockedEnd(_tokenIds[index]);
183     if (_currentLockEnd > _lockEnd) _lockEnd = _currentLockEnd;
184
185     unchecked {
```

Example (Cont.):

```sol
        ++index;
    }
}
```

Recommendation:

We advise the code to evaluate the updated lock end at the end of the function instead as a merge operation will consistently update the lock with the end time that is furthest in the future.

Alleviation (afe64e12363024e853cf5e0174506e03182346a3):

The improper `_lockEnd` assignment was relocated and corrected, evaluating the lock end of the last `zeroLockedTokenId` thus alleviating this exhibit.

Accumulator Code Style Findings

ARO-01C: Redundant Self-Transfers

Type	Severity	Location
Gas Optimization	Informational	**Accumulator.sol**: • I-1: **L62-L67** • I-2: **L73-L78**

Description:

The referenced Safe module self-transfers executed are redundant as they represent a transfer to the contract itself.

Example:

packages/lockers/src/linea/zerolend/Accumulator.sol

SOL

```sol
ILocker(locker).execTransactionFromModuleReturnData(
    ZERO,
    0,
    abi.encodeWithSelector(IERC20.transfer.selector, address(this), zeroReward),
    Enum.Operation.Call
);
```

Recommendation:

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

Alleviation (afe64e12363024e853cf5e0174506e03182346a3):

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

BaseAccumulator Code Style Findings

BAR-01C: Empty Implementation of Virtual Function

Type	Severity	Location
Code Style	Informational	BaseAccumulator.sol:L117

Description:

The referenced function implementation is redundant as a `virtual` function can exist undefined.

Example:

packages/lockers/src/common/accumulator/BaseAccumulator.sol

SOL

```sol
116 /// @notice Claims all rewards tokens for the locker and notify them to the LGV4
117 function claimAndNotifyAll(bool notifySDT, bool claimFeeStrategy) external virtual
{}
```

Recommendation:

We advise the function brackets to be replaced by a line terminal character (`;`), increasing the contract's standardization.

Alleviation (afe64e12363024e853cf5e0174506e03182346a3):

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

BAR-02C: Inefficient Data Structure

Type	Severity	Location
Gas Optimization	Informational	BaseAccumulator.sol:L25, L26

Description:

The `Split` data structure is significantly inefficient as it will utilize two distinct arrays that grow simultaneously as well as two distinct 256-bit data entries for storing the receivers and their fee numerator.

As the maximum fee numerator (`1e18`) fits within the `uint96` data type, it is possible to utilize a single 32-byte slot to store the `receiver` as well as their `fee` numerator significantly optimizing the code's gas.

Example:

packages/lockers/src/common/accumulator/BaseAccumulator.sol

SOL

```sol
/// @notice Denominator for fixed point math.
uint256 public constant DENOMINATOR = 1e18;

/// @notice Split struct
/// @param receivers Array of receivers
/// @param fees Array of fees
/// @dev First go to the first receiver, then the second, and so on
struct Split {
    address[] receivers;
    uint256[] fees; // Fee in basis points with 1e18 precision
```

Example (Cont.):

```sol
}

/// @notice Fee split.
Split feeSplit;
```

Recommendation:

We advise the `Split` structure to be set as an `address receiver` coupled with a `uint96 fee`, and the `feeSplit` contract-level variable (as well as any other code that requires an update) to be set as a `Split[]` (i.e. `Split` array) significantly optimizing the code's gas cost across several functions.

Alleviation (afe64e12363024e853cf5e0174506e03182346a3):

The structure was optimized as advised, greatly reducing the gas cost utilized by the contract.

BAR-03C: Suboptimal Struct Declaration Style

Type	Severity	Location
Code Style	Informational	BaseAccumulator.sol:L245

Description:

The linked declaration style of a struct is using index-based argument initialization.

Example:

packages/lockers/src/common/accumulator/BaseAccumulator.sol

SOL

```
245 feeSplit = Split(receivers, fees);
```

Recommendation:

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

Alleviation (afe64e12363024e853cf5e0174506e03182346a3):

The key-value declaration style is now properly in use within the referenced `struct` declaration, addressing this exhibit in full.

BaseDepositor Code Style Findings

BDR-01C: Generic Typographic Mistake

Type	Severity	Location
Code Style	Informational	BaseDepositor.sol:L139

Description:

The referenced line contains a typographical mistake (i.e. `private` variable without an underscore prefix) or generic documentational error (i.e. copy-paste) that should be corrected.

Example:

packages/lockers/src/common/depositor/BaseDepositor.sol

```sol
139 /// Transfer tokens to the locker contract and lock them.
```

Recommendation:

We advise this to be done so to enhance the legibility of the codebase.

Alleviation (afe64e12363024e853cf5e0174506e03182346a3):

The incorrect documentation line was revised to properly illustrate the ensuing line's purpose.

BDR-02C: Potential Increase of Capability

Type	Severity	Location
Standard Conformity	Informational	BaseDepositor.sol:L166, L180

Description:

The `BaseDepositor::lockToken` function does not permit a user to claim incentive tokens and stake them in the same transaction in contrast to the `BaseDepositor::deposit` function.

Example:

packages/lockers/src/common/depositor/BaseDepositor.sol

SOL

```sol
164 /// @notice Lock tokens held by the contract
165 /// @dev The contract must have Token to lock
166 function lockToken() external {
167     uint256 tokenBalance = IERC20(token).balanceOf(address(this));
168 
169     if (tokenBalance != 0) {
170         /// Transfer tokens to the locker contract and lock them.
171         SafeTransferLib.safeTransfer(token, locker, tokenBalance);
172 
173         /// Lock the amount sent.
```

Example (Cont.):

SOL

```
174         _lockToken(tokenBalance);
175     }
176
177     /// If there is incentive available give it to the user calling lockToken.
178     if (incentiveToken != 0) {
179         /// Mint incentiveToken to msg.sender.
180         ITokenMinter(minter).mint(msg.sender, incentiveToken);
181
182         /// Reset incentiveToken.
183         incentiveToken = 0;
184     }
185 }
```

Recommendation:

We advise the code of the `BaseDepositor::lockToken` function to be updated as well to support this feature, increasing the usability of the contract.

Alleviation (afe64e12363024e853cf5e0174506e03182346a3):

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

Depositor Code Style Findings

DRO-01C: Unused Error Declaration

Type	Severity	Location
Code Style	Informational	Depositor.sol:L35

Description:

The referenced `error` declaration remains unused.

Example:

packages/lockers/src/linea/zerolend/Depositor.sol

SOL

```sol
35  error ZeroLockDuration();
```

Recommendation:

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

Alleviation (afe64e12363024e853cf5e0174506e03182346a3):

The unused error declaration was renamed and re-purposed for the alleviation of `DRO-01S` thereby addressing this exhibit.

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.