Quantstamp

Lamina1 - Smart Contract Audit

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	L1 Token distribution system
Timeline	2024-04-08 through 2024-04-24
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Whitepaper
Source Code	• lamina1/rewards #659a0f7 • lamina1/airdrop-contract #05584d1
Auditors	• Nikita Belenkov Auditing Engineer • Hytham Farah Auditing Engineer • Valerian Callens Senior Auditing Engineer

Documentation quality	Low
Test quality	Low
Total Findings	44 Fixed: 22 Acknowledged: 18 Mitigated: 4
High severity findings	2 Fixed: 2
Medium severity findings	2 Fixed: 1 Acknowledged: 1
Low severity findings	20 Fixed: 11 Acknowledged: 6 Mitigated: 3
Undetermined severity findings	6 Fixed: 2 Acknowledged: 4
Informational findings	14 Fixed: 6 Acknowledged: 7 Mitigated: 1

Summary of Findings

In this audit, we reviewed Lamina1's implementation of token distribution contracts along with a reward-based voting system. When the Lamina chain goes live, the tokens will be locked in `RL1` and `LL1` contracts and released over time to users. Users can then use those newly minted `L1` tokens to vote and receive rewards. We have also reviewed the `ClaimableAirdrop` contract that would be used to airdrop `Reward` tokens to active users.

We have identified 44 issues during the course of the audit. We highly recommend that all issues be addressed before deployment. The key issues include LAM-1, which leads to double accounting at the `withdraw()`, which could ultimately result in a Denial of Service of the contract in the long run, and the issues surrounding the `Rewards` contract.

In the `Rewards` contract, we have found multiple issues (LAM-3, LAM-5, LAM-6, LAM-7, LAM-8, LAM-26, LAM-32, LAM-41 being the core ones) ranging from double voting to the possibility of Sybil attacks by well-capitalized actors on the voting system.

We also had concerns about the incentives for users to perform certain actions, such as voting (see issue LAM-42). We highly recommend doing game theory simulations to ensure that the protocol is behaving correctly and all available actions make sense from an economic perspective.

In the `ClaimableAirdrop` contract, key issues were unclear accounting of the claimed tokens and potential Merkle tree collisions between intermediate nodes and leaves.

The testing coverage should also be improved, such as having branch coverage of above 90% on each contract. End-to-end testing should also be added to the test suite.

Fix review update

All issues were addressed (fixed, mitigated, or acknowledged). The test coverage was improved, except for the `Vote` contract.

The fix review focuses only on the fixes for identified issues and, due to the significant restructuring of the codebase, cannot fully account for any new potential issues that were introduced. Hence, we recommend conducting a second audit.

ID	DESCRIPTION	SEVERITY	STATUS
LAM-1	Double Accounting Update and Event Emission when Withdrawing From `LL1`	High	Fixed
LAM-2	Accounting Inconsistencies in the Contract	High	Fixed
LAM-3	No limitations on when `distributeRewards()` can be called	Medium	Acknowledged
LAM-4	Potential Merkle Tree Collisions	Medium	Fixed
LAM-5	Vote System Vulnerable to Sybil Attack of Owners of Majority of Tokens	Low	Acknowledged
LAM-6	The `Rewards` Contract Can Be a Rewards Recipient, Hence Locking One of the Slots	Low	Fixed
LAM-7	Users Can Vote for Duplicate Recipients	Low	Fixed
LAM-8	Owner Can Bypass the Voting Process for Rewards to Promote Any Address as a First Rewards Recipient	Low	Fixed
LAM-9	Functions Vulnerable to Out-of-Gas Errors	Low	Mitigated
LAM-10	Allowance Double-Spend Exploit	Low	Acknowledged
LAM-11	Loss of Precision Due to Division before Multiplication	Low	Fixed
LAM-12	`nextClaimable()` Will Always Return `0` if at Least One Locker Has Been Unlocked	Low	Fixed
LAM-13	Unsafe Downcasting	Low	Mitigated
LAM-14	More than the Claimable Amount Can Be Claimed Due to an Incorrect Calculation in `percentOfPeriod()` for a Small Amount of `secondsPerPeriod`	Low	Acknowledged
LAM-15	Rates in `cumulativeUnlockRates` Can Reach a Rate Higher than `100%`	Low	Acknowledged
LAM-16	CEI Pattern Not Enforced in the Functions `withdraw()` and `_withdraw()`	Low	Fixed
LAM-17	Missing Input Validation	Low	Mitigated
LAM-18	`.transfer()` Should Not Be Used	Low	Fixed
LAM-19	Race Conditions Exist Between Certain Sets of Functions	Low	Acknowledged
LAM-20	The Return Value of `transfer()` Is Not Checked	Low	Fixed
LAM-21	Risk of Reentrancies	Low	Fixed
LAM-22	Revoking a Reward for a Non-Existing Node Could Disrupt the Accounting of the Contract	Low	Fixed
LAM-23	Risk of Gas Griefing Against the `owner`	Low	Acknowledged
LAM-24	Separation of Power Is Recommended	Low	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
LAM-25	Rewards Recipients with the Same Balance Are Ordered by the Date of Insertion and the Ordering Can Be Updated at No Cost	Informational	Acknowledged
LAM-26	Incentives to Collude Exist During Voting	Informational	Acknowledged
LAM-27	Application Monitoring Can Be Improved by Emitting More Events	Informational	Acknowledged
LAM-28	Unlocked Pragma	Informational	Fixed
LAM-29	The `deposit()` Function Should Be Access Controlled to Prevent Accidental Locking of Funds	Informational	Mitigated
LAM-30	Any User Can Trigger the Emission of `OMFMA` Events	Informational	Fixed
LAM-31	It Costs `27.4 L1` Tokens to Make an Address Getting More Rewards than the `Votetoken`	Informational	Fixed
LAM-32	Arbitrary Users Can Slightly Reduce the Maximum Amount of `RL1` that Can Be Minted	Informational	Fixed
LAM-33	Dead Code	Informational	Acknowledged
LAM-34	Commented-out Code	Informational	Fixed
LAM-35	`getLinkedRewards()` Returns Both Claimed and Unclaimed Rewards	Informational	Acknowledged
LAM-36	Risks Related to Using a Merkle Tree	Informational	Acknowledged
LAM-37	An Exact and Exhaustive List of Operations Allowed per Airdrop State Could Be Done to Make the System More Auditable	Informational	Fixed
LAM-38	Ownership Can Be Renounced	Informational	Acknowledged
LAM-39	Usage of Distinct Reentrancy Guards Within the Same Contract	Undetermined	Fixed
LAM-40	Updates of Unlocking and Mining Rates Can Be Done without the Existence of a Contract `omfmaRules`	Undetermined	Acknowledged
LAM-41	Unclear Behavior of the Functions `Vote.findRL1DepositsAsOf()` and `Vote.findVoteBalanceInfoAsOf()` Should Be Clarified	Undetermined	Fixed
LAM-42	Unclear incentives to vote	Undetermined	Acknowledged
LAM-43	The Mechanism of Drips and Subsidy Is Unclear	Undetermined	Acknowledged
LAM-44	Specific Aspects of the Airdrop Claiming Mechanism Should Be Clarified	Undetermined	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

The following 4 files were in scope in the `rewards` repo at commit `659a0f7fc1b47e9e59d3fc035354e28f653ab788`: `LL1.sol, RL1.sol, Vote.sol, Rewards.sol`.

The following file was in scope in the `airdrop-contract` repo at commit `05584d1c5989f8241c303bfa776c89a11cda33e6`: `ClaimableAirdrop.sol`.

Findings

LAM-1
Double Accounting Update and Event Emission when Withdrawing From `LL1`

High | Fixed

Update

The issue has been fixed by the client.
Addressed in: `0381c1eeb991125dd124de76b3ac4590b8132a00`.
The client provided the following explanation:

```
Fixed in large commit, relevant lines:
  https://github.com/lamina1/rewards/commit/0381c1eeb991125dd124de76b3ac4590b8132a00#diff-
e84dca2c1f50e6fb20502aa107eb6ca54fdb12fa6ccd437f2119732247479c6cL352
```

File(s) affected: `contracts/LL1.sol`

Description: Calling the function `withdraw()` triggers an internal call to `_withdraw()`. Both functions share the following lines:

```
totalSupply -= wad;
withdrawnTokens += wad;
```

```
lastWithdrawn = block.timestamp;
emit Withdrawal(guy, wad);
```

The impact is severe, since it will impact any on-chain actor using these accounting variables, as well as any off-chain actor monitoring the events `Withdrawal`. Moreover, this would cause a DoS by the underflow of `totalSupply -= wad` once the funds fall below a certain amount.

Recommendation: Remove the duplicated lines from the function `withdraw()`.

LAM-2 Accounting Inconsistencies in the Contract

High Fixed

Update

The issue has been fixed by the client.
Addressed in: `709bb4baf92ba7adfc918bebc40269eec83377c9`.

The client addressed all issues.

File(s) affected: `contracts/ClaimableAirdrop.sol`

Description: There are three levels of accounting in the `ClaimableAirdrop` contract (contract, airdrop, and reward level). Also, there are pairs of accounting variables where an invariant should be preserved to prevent overflow/underflow errors. For instance: `totalAllocation >= totalClaimed` and `totalSubsidy >= totalSubsidized`. Here is the list of sensitive operations where a doubt exists in terms of the correct condition to check:

1. In `createAirdrop()`, the condition `rewardToken.balanceOf(address(this)) >= allocation + totalAllocation` may be replaced with `rewardToken.balanceOf(address(this)) >= allocation + totalAllocation - totalClaimed`;
2. In `unlockAirdrop()`, the condition `rewardToken.balanceOf(address(this)) >= (airdrop.status.allocation - airdrop.status.amountClaimed) + totalAllocation` may be replaced with `rewardToken.balanceOf(address(this)) >= (airdrop.status.allocation - airdrop.status.amountClaimed) + totalAllocation - totalClaimed`;
3. In `setAllocation()`:
   - the lines `airdrop.status.allocation = newAllocation` should lead to an error if `newAllocation < airdrop.status.amountClaimed`;
   - the line `totalAllocation -= allocationDifference` should lead to an error if `newTotalAllocation - allocationDifferent < totalClaimed`;
   - the condition `rewardToken.balanceOf(address(this)) >= allocationDifference + totalAllocation` may be replaced with `rewardToken.balanceOf(address(this)) >= allocationDifference + totalAllocation - totalClaimed`;
4. In `setNumRewards()`:
   - the lines `airdrop.status.numRewards = newNumRewards` should lead to an error if `newNumRewards * status.subsidy < status.amountSubsidized`;
   - the line `totalSubsidy -= subsidyDiff` should lead to an error if `totalSubsidy - subsidyDiff < totalSubsidized`;
   - the condition `address(this).balance >= subsidyDiff + totalSubsidy` may be replaced with `address(this).balance >= subsidyDiff + totalSubsidy - totalSubsidized`;
5. In `withdrawRewardTokens()` the condition `rewardToken.balanceOf(address(this)) - totalAllocation >= amount` may be replaced with `rewardToken.balanceOf(address(this)) - (totalAllocation - totalClaimed) >= amount`;

Recommendation: Consider double-checking the different accounting updates listed above, define invariants, and update the code so that these invariants are not broken when performing an operation that updates the accounting.

LAM-3 No limitations on when `distributeRewards()` can be called

Medium Acknowledged

Update

The issue has been acknowledged by the client.
The client provided the following explanation:

```
Acknowledged, accepted risk
```

File(s) affected: `contracts/Rewards.sol`

Description: There is no limitation on the number of times `distributeRewards()` can be called. The function is only bounded by the funds that are available in the contract.

The `distributeRewards()` can also be called in the same block as `vote()`, which, if combined with the race condition outlined in LAM-19, can lead to a malicious actor observing the mempool for a deposit into the `Rewards` contract, then voting to put themselves in the top bracket and then calling `distributeRewards()` straight away.

Recommendation: Consider limiting when `distributeRewards()` can be called. One solution would be to have the `distributeRewards()` function consider the vote balance at block `n-1` when distributing rewards at block `n` so that a user cannot call `vote()` and `distributeRewards()` in the same block.

LAM-4 Potential Merkle Tree Collisions

Medium | Fixed

> **Update**
> The issue has been fixed by the client.
> Addressed in: `7218909724960aed4abfa49ab5b058b47f09b81d` .

File(s) affected: `contracts/ClaimableAirdrop.sol`

Description: The `MerkleProof` library that OpenZeppelin has developed has an edge case: an internal node in the Merkle tree could be reinterpreted as a leaf value if the leaf values are `64` bytes long prior to hashing.

The `ClaimableAirdrop` contract uses a leaf value of `recipientHashes` and `amounts`, which are `bytes32` and `uint256`, which add up to `64` bytes.

This issue is made more likely to occur due to this being an airdrop contract, and hence the users have slight control over what goes into the Merkle tree, so it makes it easier to find such non-leaf collisions.

See this nice writeup of this general issue.

Recommendation: The system should differentiate between leaf and non-leaf nodes to avoid this attack.

LAM-5
Vote System Vulnerable to Sybil Attack of Owners of Majority of Tokens

Low | Acknowledged

> **Update**
> The issue has been acknowledged by the client.
> The client provided the following explanation:
>
> ```
> Acknowledged, accepted risk
> ```

File(s) affected: `contracts/Rewards.sol`

Description: An actor with a lot of funds may be incentivized to split their recipient contracts multiple times in order to receive a greater number of rewards. This would be achieved by effectively kicking out what would have been the last address appearing in the call to the `sortedAccounts()` function.

Considered unlikely since only an address with a large amount of tokens can do that attack (around `80%` of the total token supply).

Exploit Scenario: To make the matter simple, assume recipient `A` knows they can get `80` votes and all other recipients will only get `5` votes. An honest actor would cast themselves as a single recipient, and the final distribution of the top five recipients would look like this: `[80, 5, 5, 5, 5]`, so that `A` gets `80%` of the total rewards. However, if they split into 5 different recipients and distributed a total of `16` votes to each recipient, the final tally would then look like `[16, 16, 16, 16, 16]`, so that `A` would get `100%` of the total rewards.

Recommendation: Consider reassessing the design of the voting system. If this risk remains, consider documenting it to end-users.

LAM-6
The `Rewards` Contract Can Be a Rewards Recipient, Hence Locking One of the Slots

Low | Fixed

> **Update**
> The issue has been fixed by the client.
> Addressed in: `0381c1eeb991125dd124de76b3ac4590b8132a00` .
> The client provided the following explanation:
>
> ```
> Fixed in large commit, relevant lines:
> https://github.com/lamina1/rewards/commit/0381c1eeb991125dd124de76b3ac4590b8132a00#diff-
> e958576ab184fa44250f57f3129af7374508b47c595196740b7516404bf331d4R92
> ```

File(s) affected: `contracts/Rewards.sol`

Description: No check makes sure that the `Rewards` contract cannot become a recipient of rewards. For this, enough users should vote to promote this `Rewards` contract as an eligible recipient. This means that the `Rewards` contract is locking a slot away from a potential valid recipient.

Then, when rewards are distributed, the `Rewards` contract will get a portion of rewards that could be dispatched among the other eligible recipients in consecutive additional rounds.

Exploit Scenario: For instance, with `Rewards.maxRecipients = 3`, the `selectedAcounts = [ (voteToken, 50), (Bob, 25), (Rewards contract, 25)]`, and `100` rewards to be distributed:

1. Bob executes `Rewards.distributeRewards()`:
   - voteToken gets `50`, Bob gets `25`, Rewards contract gets `25`;
   - Rewards contract still has `25` tokens;
1. Bob executes `Rewards.distributeRewards()`:
   - voteToken gets `12`, Bob gets `6`, Rewards contract gets `6`;
   - Rewards contract still have `7` tokens;
1. Bob executes `Rewards.distributeRewards()`:
   - voteToken gets `3`, Bob gets `1`, Rewards contract gets `1`;
   - Rewards contract still have `3` tokens; At the end:
   - voteToken received `65` tokens;
   - Bob received `32` tokens, instead of `25`;

Further simulations should be done to assess if, from a game theory perspective, this flaw could be economically profitable. As a result, the exact likelihood of this scenario was not explored in detail.

Recommendation: Add a check to make sure that the `Rewards` contract cannot be voted on via the function `vote()`.

LAM-7 Users Can Vote for Duplicate Recipients

Low | Fixed

Update

The issue has been fixed by the client.
Addressed in: `0381c1eeb991125dd124de76b3ac4590b8132a00`.
The client provided the following explanation:

```
Fixed in large commit, relevant lines:
  https://github.com/lamina1/rewards/commit/0381c1eeb991125dd124de76b3ac4590b8132a00#diff-
e958576ab184fa44250f57f3129af7374508b47c595196740b7516404bf331d4R95
```

File(s) affected: `contracts/Rewards.sol`

Description: In the `Rewards.vote()` function, a user may put the same recipient multiple times, allowing them to generate a large positive vote in their favour or a large negative vote against them.

Recommendation: Consider alternatively:

1. Mitigating the issue by ensuring the sum of the absolute value of the scores is less than or equal to `100`.
2. Fixing the issue by adding an initial for loop, making sure that addresses are strictly ordered in an ascending order, and reverting if it is not the case.

LAM-8
Owner Can Bypass the Voting Process for Rewards to Promote Any Address as a First Rewards Recipient

Low | Fixed

Update

The issue has been fixed by the client.
Addressed in: `0381c1eeb991125dd124de76b3ac4590b8132a00`.
The client provided the following explanation:

```
Fixed in large commit, relevant lines:
  https://github.com/lamina1/rewards/commit/0381c1eeb991125dd124de76b3ac4590b8132a00#diff-
e958576ab184fa44250f57f3129af7374508b47c595196740b7516404bf331d4L45
```

File(s) affected: `contracts/Rewards.sol`

Description: The privileged address `Rewards.owner` can set an address as the `voteToken` via the function `Rewards.setVoteToken()`. It results in this address getting inserted in the `AccountStorage` tree with the balance `100 * 1000 ether` to make it first in the order of rewards recipients. As a result, the following steps can be used to make the address of Alice a recipient:

1. `owner` calls `setVoteToken()` with the legitimate `voteToken` address.
2. `owner` calls `setVoteToken()` with the address of `Alice`.
3. `owner` calls `setVoteToken()` with the legitimate `voteToken` address. As a result, the list of sorted accounts will return `[(address: voteToken, balance: 100k ethers),(address: Alice, balance: 100k ethers)]`. The likelihood is Low since the address `Rewards.owner` is considered trusted. But if this happens, it will not be easy to mitigate the impact of this operation, and it will have a large impact.

Recommendation: Depending on the expected behavior, prevent adding twice a `voteToken` or reduce the current balance of any existing `voteToken` to `0` before setting a new address as the `voteToken`.

LAM-9 Functions Vulnerable to Out-of-Gas Errors

• Low Mitigated

Update

The issue has been mitigated by the client.

Addressed in: `0381c1eeb991125dd124de76b3ac4590b8132a00`.
The client provided the following explanation:

```
Changes to Vote contract mitigate these concerns
```

File(s) affected: `contracts/Vote.sol`, `contracts/LL1.sol`, `contracts/RL1.sol`

Description: At several locations, a loop iterates over a list. However, if the size of the list becomes too big, the amount of gas to perform this operation may lead to an out-of-gas error, which could lead to a temporary or complete denial of service for this function. It is the case, for instance, in:

1. In `Vote.getVoteBalanceChanges()`;
2. In `Vote.getRL1Deposits()`;
3. In `Vote.claimAll()`;
4. In `Vote.totalLocked()`;
5. In `Vote.totalClaimable()`;
6. In `Vote.unlockable()`;
7. In `Vote.unlockRange()`;
8. In `LL1.ensureProperRatePeriods()`;
9. In `LL1.claimable()`;
10. In `RL1.claimable()`;

Recommendation: Consider refactoring the functions to make sure the likelihood of an out-of-gas error is reduced. Especially for `Vote.unlockRange()`, a check in `Vote.unlock()` could make sure that we only execute the body of the function if `locker[msg.sender][index].amount != 0`.

LAM-10 Allowance Double-Spend Exploit

• Low Acknowledged

Update

The issue has been acknowledged by the client.
The client provided the following explanation:

```
All ERC20s have same issue, dealt with on wallet side
```

File(s) affected: `contracts/LL1.sol`, `contracts/RL1.sol`

Description: As it presently is constructed, the contract is vulnerable to the allowance double-spend exploit, as with other ERC20 tokens.

The affected functions in the case of `RL1` and `LL1` tokens would be :

- `RL1.approve()`
- `LL1.approve()`

Exploit Scenario: 1. Alice allows Bob to transfer `N` amount of Alice's tokens (`N>0`) by calling the `approve()` method on `Token` smart contract (passing Bob's address and `N` as method arguments)

2. After some time, Alice decides to change from `N` to `M` (`M>0`) the number of Alice's tokens Bob is allowed to transfer, so she calls the `approve()` method again, this time passing Bob's address and `M` as method arguments
3. Bob notices Alice's second transaction before it was mined and quickly sends another transaction that calls the `transferFrom()` method to transfer `N` Alice's tokens somewhere

4. If Bob's transaction will be executed before Alice's transaction, then Bob will successfully transfer `N` Alice's tokens and will gain the ability to transfer another `M` tokens
5. Before Alice notices any irregularities, Bob calls the `transferFrom()` method again, this time to transfer `M` Alice's tokens.

Recommendation: The exploit (as described above) is mitigated through the use of functions that increase/decrease the allowance relative to its current value, such as `increaseAllowance()` and `decreaseAllowance()`. Furthermore, we recommend that developers of applications dependent on `approve()` / `transferFrom()` should keep in mind that they have to set the allowance to `0` first and verify if it was used before setting the new value.

LAM-11 Loss of Precision Due to Division before Multiplication

Low Fixed

Update

The issue has been fixed by the client.
Addressed in: `0381c1eeb991125dd124de76b3ac4590b8132a00`.
The client provided the following explanation:

```
Fixed in large commit, relevant lines:
  LL1: https://github.com/lamina1/rewards/commit/0381c1eeb991125dd124de76b3ac4590b8132a00#diff-
e84dca2c1f50e6fb20502aa107eb6ca54fdb12fa6ccd437f2119732247479c6cR413
  RL1: https://github.com/lamina1/rewards/commit/0381c1eeb991125dd124de76b3ac4590b8132a00#diff-
08e835fb020057e51897bc2b9b4de5e6d15a821ec941b1f578262f9840e358daR576
```

File(s) affected: `contracts/LL1.sol`, `contracts/RL1.sol`

Description: Division before multiplication may result in a loss of precision when the operations are carried over integer numbers. This is the case in the following functions:

1. Function `RL1.transferFrom()` performs a multiplication on the result of a division:
   - `shareOfClaimed = (totalWithdrawn[src] * 10000) / totalMinted[src]`
   - `claimAmtToMove = (wad * shareOfClaimed) / 10000`
2. Function `LL1.transferFrom()` performs a multiplication on the result of a division:
   - `pctWithdrawn = (totalWithdrawn[src] * 10000) / totalMinted[src]`
   - `withdrawAmtToMove = (wad * pctWithdrawn) / 10000`

Recommendation: Rewrite operations so that the division happens after multiplication.

LAM-12 `nextClaimable()` Will Always Return `0` if at Least One Locker Has Been Unlocked

Low Fixed

Update

The issue has been mitigated by the client.
Addressed in: `0381c1eeb991125dd124de76b3ac4590b8132a00`.
The client provided the following explanation:

```
Changes to Vote contract mitigate these concerns
```

File(s) affected: `contracts/Vote.sol`

Description: The `nextClaimable()` function is supposed to return the index of the next claimable lock for an address. This is done by iterating the `locker` mapping and returning the lowest `unlockTime` value, but this also covers the deleted lockers that have `0` as `unlockTime`, which means that instead this function will return the latest unlocked locker as the index instead of the next claimable.

Recommendation: Make sure that the deleted lockers are skipped instead of being accounted for.

LAM-13 Unsafe Downcasting

Low Mitigated

Update

The issue has been mitigated by the client.
Addressed in: `0381c1eeb991125dd124de76b3ac4590b8132a00`.
The client provided the following explanation:

```
Fixed some of downcasts and left comments on values that are properly scoped
```

File(s) affected: `contracts/Vote.sol`, `contracts/RL1.sol`, `contracts/Rewards.sol`

Description: At several locations, a higher type such as `uint256` is unsafely downcasted to a lower type such as `uint16`. As a result, any truncation of the initial value would pass without any error, which could lead to an unexpected outcome. It is the case:

1. `LL1`:
    1. `rateLength()`,
    2. `periodFor()`,
    3. `currentPeriod()`,
    4. `percentOfPeriod()`,
    5. `percentOfCurrentPeriod()`.
2. `RL1`:
    1. `rateLength()`,
    2. `periodFor()`,
    3. `currentPeriod()`,
    4. `percentOfPeriod()`,
    5. `percentOfCurrentPeriod()`.
3. `Vote`:
    1. `today()`,
    2. `dayAt()`.
4. In `Rewards.distributeRewards()`, in the for loop. We suggest for this one to declare and store `maxRecipients` with the type `uint8`;
5. In `Rewards.vote()`, the `numVoteTokens` variable is passed in as a `uint` but then later cast to an `int`. The issue arises from the fact that `int` and `uint` do represent slightly different ranges of values, as they use one bit to represent the 2s complement encoding of negative values. Hence, if `numVoteTokens` is large enough, it can be cast to a large negative number due to the 2s complement being treated incorrectly.

Recommendation: We recommend using the `SafeCast` library of OpenZeppelin to fail if the initial value is truncated when downcasted.

LAM-14

More than the Claimable Amount Can Be Claimed Due to an Incorrect Calculation in `percentOfPeriod()` for a Small Amount of `secondsPerPeriod`

Low | Acknowledged

Update

The issue has been acknowledged by the client.
The client provided the following explanation:

```
Function is unused
```

File(s) affected: `contracts/RL1.sol`, `contracts/LL1.sol`

Description: In the contract `RL1`, the function `percentOfPeriod()` is used by the function `claimable()`, initially used by the function `claim()`. Let's consider a simplified scenario where `startDate = 10` and `secondsPerPeriod = 3` (note that this can happen if `debugSpeedup = 2628000`). In this situation, period `0` should contain the timestamps `[10,11,12]` and period `1` the timestamps `[13,14,15]`. The function `claimable()` will call the function `percentOfPeriod()` with two timestamps: `lastClaimed` which should be included in the period, and `block.timestamp`, which should be excluded from the period. However, `periodStartSeconds(n)` returns the first timestamp included in the period `n`, and `periodEndSeconds(n)` the timestamp after the last timestamp of the period `n` (which should not be counted in the period `n`, because it is the first timestamp of period `n+1`). In our simple scenario, calling `percentOfPeriod(0,10,13)` at timestamp `13` will return `(13 - 10 + 1) * 10000 / 3 = 40000 / 3 = 13333`, which corresponds to a percentage of `133%` of the claimable amount of period `0`, instead of `100%`. The impact is limited to disrupting the test scenarios because the value of `debugSpeedup` must be high to increase the incorrect percentage. In a normal scenario where `startDate = 10` and `secondsPerPeriod = 7884000`, calling `percentOfPeriod(0,10,7884013)` at timestamp `7884013` will return the value `10000`, which corresponds to a percentage of `100%`. However, this corresponds to `10000.005` and not `10000.000`.

Recommendation: 1. Replace in the function `percentOfPeriod()` of both contracts:

```
uint32((end - start + 1) * 10000 / secondsPerPeriod);
```

with

```
uint32((end - start) * 10000 / secondsPerPeriod);
```

2. Make sure with robust and exhaustive tests that the root cause of this issue is now fixed. In the simplified scenario, we should get `3333` when we call `percentOfPeriod(0,10,11)`, because only the timestamp `10` out of the timestamps `[10,11,12]` is considered valid.

LAM-15
Rates in `cumulativeUnlockRates` Can Reach a Rate Higher than `100%`

Low | Acknowledged

> **Update**
> The issue has been acknowledged by the client.
> The client provided the following explanation:
>
> `Function is privileged, sending account is trusted to only set valid values`

File(s) affected: `contracts/LL1.sol`, `contracts/RL1.sol`

Description: In the contracts `LL1` and `RL1`, the list `cumulativeUnlockRates` is updated by inserting at the end the new value added to the value of its current last element. No check makes sure in the code that the inserted value cannot be more than `100%`.

Recommendation: Consider making sure that the maximum value that can be inserted into `cumulativeUnlockRates` is `100%`.

LAM-16
CEI Pattern Not Enforced in the Functions `withdraw()` and `_withdraw()`

Low | Fixed

> **Update**
> The issue has been fixed by the client.
> Addressed in: `0381c1eeb991125dd124de76b3ac4590b8132a00`.
> The client provided the following explanation:
>
> ```
> Fixed in large commit, relevant lines:
>  https://github.com/lamina1/rewards/commit/0381c1eeb991125dd124de76b3ac4590b8132a00#diff-
> e84dca2c1f50e6fb20502aa107eb6ca54fdb12fa6ccd437f2119732247479c6cR350
> ```

File(s) affected: `contracts/LL1.sol`

Description: In the functions `withdraw()` and `_withdraw()`, storage variables are updated after a `transfer()`. This is against the CEI pattern (Checks Effects Interactions) and could lead to specific cases of reentrancy (read-only or non-read-only ones).

Recommendation: Enforce the CEI pattern.

LAM-17 Missing Input Validation

Low | Mitigated

> **Update**
> The issue has been fixed by the client.
> Addressed in: `4f370616928bb86cec0b3bb86189a519a4a83899`.

File(s) affected: `contracts/Vote.sol`, `contracts/Rewards.sol`, `contracts/RL1.sol`, `contracts/LL1.sol`, `contracts/ClaimableAirdrop.sol`

Related Issue(s): SWC-123

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error:

1. In `LL1.sol` and `RR1.sol`: a. In the `constructor()`:
   - `_pS` can be any timestamp in the past or the future;
   - `_sPP` can be any amount starting from `1`;
   - `rewards` can be an `address(0)`;
1. In `Vote.sol`: a. In `setStartDay()`, `_startDayEpochSeconds` can be in the future, in the past and not match `GMT 00:00` (meaning `% secondsPerDay != 0`);
2. In `Rewards.sol`: a. In `setMaxRecipients()`, there is no max value check for `_maxRecipients`; b. In `vote()`, `numVoteTokens` and the items in `_scores` can be `0`.
3. In `RL1.sol`: a. In `setRewardsContract()`, `dst` can be an `address(0)`.
4. In `ClaimableAirdrop.sol`: a. In `claimRewards()`, `recipients.length` is not checked; b. In `setAllocation()`, `newAllocation` can be `0`; c. In `setNumRewards()`, `newNumRewards` can be `0`.

Recommendation: We recommend adding the relevant checks.

LAM-18 `.transfer()` Should Not Be Used

• Low Fixed

Update

The issue has been fixed by the client.
Addressed in: `0381c1eeb991125dd124de76b3ac4590b8132a00`.
The client provided the following explanation:

```
Fixed in large commit, relevant lines:
  https://github.com/lamina1/rewards/commit/0381c1eeb991125dd124de76b3ac4590b8132a00#diff-
e84dca2c1f50e6fb20502aa107eb6ca54fdb12fa6ccd437f2119732247479c6cR364
```

File(s) affected: `contracts/RL1.sol`, `contracts/LL1.sol`, `contracts/Vote.sol`

Description: Solidity's `.transfer()` function forwards a fixed amount of 2300 gas to the recipient when transferring native tokens to them, which causes several limitations if the recipient is a contract. First, the limited amount of gas prevents the contract from executing more complicated logic (e.g., external calls) in its `fallback()` or `receive()` function. Second, the gas costs of each opcode are subject to change; therefore, it is not guaranteed that currently successful transfers will not fail in the future. The following functions in the listed contract use `.transfer()` to transfer native tokens:

1. `RL1._withdrawFor()`
2. `LL1._withdrawFor()`
3. `Vote.unlock()`

Recommendation: Consider using `.call{value: value_}("")` to transfer native tokens to the recipient. This fix needs to be carefully introduced together with LAM-16 to avoid potential reentrancy attack vectors.

LAM-19
Race Conditions Exist Between Certain Sets of Functions

• Low Acknowledged

Update

The issue has been acknowledged by the client.
The client provided the following explanation:

```
Working as intended
```

File(s) affected: `contracts/Rewards.sol`, `contracts/RL1.sol`

Description: The process of distributing rewards depends on the order of eligible addresses in the `AccountStorage` tree and the maximum number of recipients. Since the following functions (`vote()`, `setMaxRecipients()`, `distributeRewards()` and `setVoteToken()`) involve these variables in a read or write access, they are all exposed to a race condition.

The process of minting RL1 depends on the amount already minted and can impact the value of the mappings `claimableProofTokens` and `cumClaimableProofTokens` for the current period. As a result, these functions are also exposed to a race condition.

Recommendation: Consider if adding a timelock mechanism to any of these functions is necessary. Also, document to end-users the fact that these functions can be front-run and that these should be executed via private RPCs.

LAM-20 The Return Value of `transfer()` Is Not Checked

• Low Fixed

Update

The issue has been fixed by the client.
Addressed in: `c0bcd43562f932d67c6297e4070553808e8bd3c4`.

File(s) affected: `contracts/ClaimableAirdrop.sol`

Description: The `withdrawRewardTokens()` and `withdrawTokens()` functions transfer tokens to a recipient, but do not check the return value if the transfer fails.

Recommendation: Consider using `SafeTransfer()` from OpenZeppelin to check the return value for both standard and non-standard ERC-20 tokens.

LAM-21 Risk of Reentrancies

• Low Fixed

Update

The issue has been fixed by the client.
Addressed in: `5f570d766077cd419dc6d7d2d8ecd052aa989fdf`.

File(s) affected: `contracts/ClaimableAirdrop.sol`

Description: Some operations can result in a transfer of native tokens via the low-level instruction `call()`: `linkAddress()` and `claimRewards()` or of reward tokens: `claimRewards()` and `claimMyRewards()`. In these functions:

1. the CEI pattern is not always enforced as storage variables are updated after the external call;
2. the nonReentrant modifier is used in the internal function `claimNode()` but not in `linkAddress()`. Even if only the owner can call `linkAddress()`, non-exhaustive respect of the CEI pattern makes the system vulnerable to different types of reentrancies (read-only or not).

These functions are also missing a `nonReentrant` modifier and make an external call:

1. `withdrawEther()`
2. `withdrawAllEther()`

Recommendation: Consider making sure that the CEI pattern is implemented everywhere it can be, and use the `nonReentrant` modifier where the CEI pattern cannot be fully implemented.

LAM-22
Revoking a Reward for a Non-Existing Node Could Disrupt the Accounting of the Contract

Low · Fixed

Update

The issue has been fixed by the client.
Addressed in: `b87fcdc1a80885d663e45c8291abb67e9c601371`.

File(s) affected: `contracts/ClaimableAirdrop.sol`

Description: The owner of the contract can revoke rewards for a given airdrop via the function `revokeRewards()`. However, if any pair of `(recipientHashes[i], amounts[i])` does not match an existing node of this airdrop, the following accounting variables will be incorrectly updated:

- `airdrop.status.allocation`;
- `totalAllocation`;
- `airdrop.status.numRewards`;
- `totalSubsidy`;

Recommendation: We recommend to revert if the pair `(recipientHashes[i], amounts[i])` does not match an existing node.

LAM-23 Risk of Gas Griefing Against the `owner`

Low · Acknowledged

Update

The issue has been acknowledged by the client.
The client provided the following explanation:

```
Acknowledged, fixed on owner bot
```

File(s) affected: `contracts/ClaimableAirdrop.sol`

Description: The `owner` of the contract can trigger transactions that will result in a transfer of native tokens via the low-level instruction `call()`: `linkAddress()` and `claimRewards()`. If the `recipient` is a smart contract, it can implement a business logic that will use a huge amount of gas.

Recommendation: Consider making sure that the `owner` simulates transactions before executing them to make sure that it will not result in an abnormally high amount of gas.

LAM-24 Separation of Power Is Recommended

Low · Fixed

Update

The issue has been fixed by the client.

File(s) affected: `contracts/ClaimableAirdrop.sol`

Description: Currently, the contract has one privileged role, the `owner`, but the system will have multiple actors that use the same role for different actions. For example, `createAirdrop()` will be called by the foundation using the `owner` account, but `linkAddress()` will be used by the Discord bot also via `owner` account. So it is a good practice to separate these 2 roles into their own to have the principle of least privilege, as the Discord bot does not need the ability to `createAirdrop()`. This would also limit the impact if the key is compromised.

Recommendation: Consider creating a new role for the Discord bot and using a multi-sig wallet for the `owner` account.

LAM-25
Rewards Recipients with the Same Balance Are Ordered by the Date of Insertion and the Ordering Can Be Updated at No Cost

Informational | Acknowledged

Update

The issue has been acknowledged by the client.
The client provided the following explanation:

```
Acknowledged, accepted risk
```

File(s) affected: `contracts/Rewards.sol`

Description: Addresses are inserted in the `AccountStorage` tree with a given balance, and ordered within the tree based on that balance. If the address of Bob is inserted into the tree with the balance `b` and the address of Alice is already in the tree with the balance `b`, then the address of Alice is considered to be "before" the address of Alice. As a result, if we consider that Charlie wants to vote equally for Alice and Bob via the function `Rewards.vote()`, using the parameters `[Alice, Bob], [50, 50]` or `[Bob, Alice], [50, 50]` will not provide the same outcome in terms of ordering. The impact gets more severe since:

- the list of eligible rewards recipients is truncated depending on the value of `Rewards.maxRecipients`.
- it is possible to call `Rewards.vote()` with `numVoteTokens == 0`, to update the ordering of `n` inserted addresses with the same balance at literally no cost.

Recommendation: Consider reassessing the design of the voting system. If this risk remains, consider documenting it to end-users.

LAM-26 Incentives to Collude Exist During Voting

Informational | Acknowledged

Update

The issue has been acknowledged by the client.
The client provided the following explanation:

```
Acknowledged, accepted risk
```

File(s) affected: `contracts/Rewards.sol`

Description: It is possible to create a forwarder contract that can receive funds and split them across several contracts. If such a forward contract is set as one of the reward recipients, they can, in effect, collude to artificially arrive to be one of the parties returned by the call to `sortedAccounts(maxRecipients)` within the `distributeRewards()` function.

Exploit Scenario: Suppose there are 2 parties with contracts they want to receive rewards from, call them `A` and `B`, who fear they will not get enough votes to be in the top five and hence receive zero rewards.

They can deploy a forwarder contract, `F`, that splits all funds received between `A` and `B` but then cast votes for `F` as the recipient contract. In effect, they will now receive the sum of the votes they each would have received individually, which would make it more likely that they will be in the top five recipients and hence receive rewards.

Recommendation: Document the risk to the users and assess the design of the voting system.

LAM-27
Application Monitoring Can Be Improved by Emitting More Events

Informational | Acknowledged

Update

The issue has been acknowledged by the client.
The client provided the following explanation:

```
Acknowledged, accepted less events for indexing
```

File(s) affected: `contracts/LL1.sol`, `contracts/RL1.sol`, `contracts/Vote.sol`, `contracts/Rewards.sol`

Description: In order to validate the proper deployment and initialization of the contracts, it is a good practice to emit events. Also, any important state transitions can be logged, which is beneficial for monitoring the contract, and also tracking eventual bugs or hacks. Below we present a non-exhaustive list of events that could be emitted to improve application management:

The following instances are missing event emissions on state changes:

1. **LL1**:
    1. `deposit()` missing events for variables: `balanceOf`, `totalDeposited`, `totalMinted`, `totalSupply`
    2. `ensureProperRatePeriods()` missing events for variables: `cumulativeUnlockRates`, `unlockRates`
    3. `withdraw()` missing events for variables: `lastWithdrawn`, `totalSupply`, `withdrawnTokens`
    4. `onlyOMFMA()` missing events for variables: `lastHeardFromOMFMA`
    5. `acceptOwnership()` missing events for variables: `owner`
    6. `removeLL1Sender()` missing events for variables: `LL1Senders`
    7. `_withdrawFor()` missing events for variables: `balanceOf`, `lastWithdrawn`, `totalSupply`, `totalWithdrawn`, `withdrawnTokens`
    8. `constructor()` missing events for variables: `debugSpeedup`, `owner`, `periodStart`, `secondsPerPeriod`
    9. `transferFrom()` missing events for variables: `allowance`, `balanceOf`, `totalMinted`, `totalWithdrawn`
    10. `setLL1Sender()` missing events for variables: `LL1Senders`
    11. `_withdrawFor()` missing events for variables: `balanceOf`, `lastWithdrawn`, `totalSupply`, `totalWithdrawn`, `withdrawnTokens`
    12. `acceptOwnership()` missing events for variables: `owner`, `pendingOwner`
    13. `transferOwnership()` missing events for variables: `pendingOwner`
    14. `withdraw()` missing events for variables: `lastWithdrawn`, `totalSupply`, `withdrawnTokens`
    15. `setOMFMARules()` missing events for variables: `omfmaRules`
    16. `setUnlockRate()` missing events for variables: `cumulativeUnlockRates`.
2. **RL1**:
    1. `transferFrom()` missing events for variables: `allowance`, `balanceOf`, `totalMinted`, `totalWithdrawn`, `transferGuard`.
    2. `mintRL1()` missing events for variables: `balanceOf`, `claimableProofTokens`, `cumClaimableProofTokens`, `totalDeposited`, `totalMinted`.
    3. `trustedTransfer()` missing events for variables: `balanceOf`, `totalMinted`.
    4. `acceptOwnership()` missing events for variables: `owner`.
    5. `setUnlockRate()` missing events for variables: `cumulativeUnlockRates`.
    6. `constructor()` missing events for variables: `debugSpeedup`, `owner`, `periodStart`, `rewardsContract`, `secondsPerPeriod`.
    7. `ensureProperReleaseRate()` missing events for variables: `claimableReleaseRate`.
    8. `setRL1Sender()` missing events for variables: `RL1Senders`.
    9. `removeRL1Sender()` missing events for variables: `RL1Senders`.
    10. `setMiningRate()` missing events for variables: `claimableProofTokens`, `claimableReleaseRate`, `cumClaimableProofTokens`.
    11. `onlyOMFMA()` missing events for variables: `lastHeardFromOMFMA`.
    12. `ensureProperRewardPeriods()` missing events for variables: `claimableProofTokens`, `cumClaimableProofTokens`.
    13. `rewardsTransfer()` missing events for variables: `balanceOf`, `totalMinted`
    14. `transferOwnership()` missing events for variables: `pendingOwner`
    15. `claim()` missing events for variables: `claimedTokens`, `lastClaimed`.
    16. `setRewardsContract()` missing events for variables: `rewardsContract.
    17. `setOMFMARules()` missing events for variables: `omfmaRules`.
3. **Rewards**:
    1. `setVoteToken()` missing events for variables: `voteToken`
    2. `setMaxRecipients()` missing events for variables: `maxRecipients`
    3. `setRl1()` missing events for variables: `rl1`
4. **Vote**:
    1. `setPaused()` missing events for variables: `paused`
    2. `lock()` missing events for variables: `lockCount`, `lockList`, `locker`, `unlockableOnDay`.
    3. `unlock()` missing events for variables: `locker`, `unlockedOnDay`.
    4. `setRL1Contract()` missing events for variables: `rl1`.
    5. `setStartDay()` missing events for variables: `rl1Deposits`, `startDay`, `voteBalanceChanges`
    6. `setStartDay()` missing events for variable: `startDay`.
    7. `_update()` missing events for variables: `voteBalanceChanges`
    8. `receiveRL1()` missing events for variable: `rl1Deposits`.
    9. `claimRL1()` missing events for variables: `claimed`
    10. `setRewardsContract()` missing events for variables: `rewards`

Recommendation: Ensure that all state changes emit an event to facilitate tracking and auditing.

LAM-28 Unlocked Pragma

Informational | Fixed

Update

The issue has been fixed by the client.
Addressed in: `0381c1eeb991125dd124de76b3ac4590b8132a00`.
The client provided the following explanation:

```
Fixed in large commit, relevant lines:
  https://github.com/lamina1/rewards/commit/0381c1eeb991125dd124de76b3ac4590b8132a00#diff-
e84dca2c1f50e6fb20502aa107eb6ca54fdb12fa6ccd437f2119732247479c6cL2
```

File(s) affected: `contracts/RL1.sol`, `contracts/LL1.sol`, `contracts/Rewards.sol`, `contracts/Vote.sol`, `contracts/ClaimableAirdrop.sol`

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend to remove the caret to lock the file onto a specific Solidity version.

LAM-29
The `deposit()` Function Should Be Access Controlled to Prevent Accidental Locking of Funds

Informational | Mitigated

Update

The issue has been mitigated by the client.
Addressed in: `0381c1eeb991125dd124de76b3ac4590b8132a00`.
The client provided the following explanation:

```
https://github.com/lamina1/rewards/commit/0381c1eeb991125dd124de76b3ac4590b8132a00#diff-
e84dca2c1f50e6fb20502aa107eb6ca54fdb12fa6ccd437f2119732247479c6cL309
```

File(s) affected: `contracts/RL1.sol`, `contracts/LL1.sol`

Description: Deposits into `LL1` and `RL1` contracts will only be done by the Lamina1 team and hence should not be accessed by normal users. It would be a good idea to add access control to such `deposit()` calls as otherwise users can accidentally lock their funds in the contract.

Recommendation: Consider adding access controls to `deposit()`

LAM-30 Any User Can Trigger the Emission of `OMFMA` Events

Informational | Fixed

Update

The issue has been fixed by the client.
Addressed in: `0381c1eeb991125dd124de76b3ac4590b8132a00`.
The client provided the following explanation:

```
Fixed in large commit, relevant lines:
  https://github.com/lamina1/rewards/commit/0381c1eeb991125dd124de76b3ac4590b8132a00#diff-
e84dca2c1f50e6fb20502aa107eb6ca54fdb12fa6ccd437f2119732247479c6cR100
```

File(s) affected: `contracts/LL1.sol`, `contracts/RL1.sol`

Description: The function `setOMFMA()` is public. As a result, any user can execute it and trigger the emission of `OMFMA` events, which could disrupt the normal operation of external observers monitoring events from the contract.

Recommendation: We recommend adding a revert statement to an additional else block if no other conditional statement passes.

LAM-31

It Costs `27.4 L1` Tokens to Make an Address Getting More Rewards than the `Votetoken`

Informational | Fixed

Update

The issue has been fixed by the client.
Addressed in: `0381c1eeb991125dd124de76b3ac4590b8132a00`.
The client provided the following explanation:

```
Fixed in large commit, relevant lines:
  https://github.com/lamina1/rewards/commit/0381c1eeb991125dd124de76b3ac4590b8132a00#diff-
e958576ab184fa44250f57f3129af7374508b47c595196740b7516404bf331d4R28
```

The client added the comment: We addressed the issue by setting initial reward recipients and their scores in the constructor of the contract. You can see the planned values in the deploy script in the latest commit: it sets 38.325 B ETH score for voting and 8.2125 B ETH score for node operators and ecosystem fund.
This means that to reach the same score as the vote contract you would need to stake a bit more than 100k L1 for 10 years, which would give you 365 M vote tokens, that can get a score of 36.5 B (by burning all the tokens)
We've recently done some more simulations of early days and this is probably fine as the needed values are quite large. However, we might need to increase these initial scores, or even change the implementation so that there's always a minimum possible percentage of rewards that gets allocated to stakers.

File(s) affected: `contracts/Rewards.sol`

Description: Users can vote to update the order in which addresses will get rewards from the contract `Rewards`. When a vote token is added as a rewards receiver (in `Rewards.setVoteToken()`), it is inserted in the `AccountStorage` tree with the balance `100 * 1000 ether` to make it first. Depending on the monetary costs in the order of rewards recipients. However, depending on the monetary cost of an `L1 token` in the future, the opportunity cost to make it first in the order of rewards recipients through votes may be limited, hence accessible to an actor with a positive interest in doing so.

Recommendation: Consider:

1. Assessing whether the vote token should be part of the list of recipients or always get a hardcoded ratio of the rewards so it is no longer part of the rewards recipients ranking.
2. If option 1 is not an option, assess if the initial value of 100k is appropriate (not too high or low).

LAM-32

Arbitrary Users Can Slightly Reduce the Maximum Amount of `RL1` that Can Be Minted

Informational | Fixed

Update

The issue has been fixed by the client.
Addressed in: `0381c1eeb991125dd124de76b3ac4590b8132a00`.
The client provided the following explanation:

```
Fixed in large commit, relevant lines:
  https://github.com/lamina1/rewards/commit/0381c1eeb991125dd124de76b3ac4590b8132a00#diff-
e958576ab184fa44250f57f3129af7374508b47c595196740b7516404bf331d4R93
```

File(s) affected: `contracts/Rewards.sol`

Description: In `RL1.mintRL1()`, the following check makes sure that the `RL1.owner` cannot mint more than `500_000_000 ether`:

```
require (totalMinted[address(this)] + wad <= 500_000_000 ether, "Already minted 500M RL1");
```

However, any address can via a vote add the `RL1` contract as a rewards recipient, which will trigger a call to `rewardsTransfer(dst: address(RL1), wad: n)` that will increase the value of `totalMinted[address(RL1)]`.

For instance:

1. `owner` mints `100M RL1` via `RL1.mintRL1(100M ethers);`
2. `Bob` votes to add `RL1` as a valid rewards recipient.
3. `Bob` calls `Rewards.distributeRewards()`, which leads to a non-zero amount `n` transferred to `address(RL1)` that will execute the operation `totalMinted[address(RL1)] += n`;
4. `owner` cannot mint `400M RL1` via `RL1.mintRL1(400M ethers);`, because now `totalMinted[address(RL1)] + 400M ethers` is greater of `500M ethers` by an amount of `n`.

Even if the incentive is not clear, making this scenario unlikely, this flaw can easily be fixed by replacing the accounting variable used in the check.

Recommendation: Update

```
require (totalMinted[address(this)] + wad <= 500_000_000 ether, "Already minted 500M RL1");
```

with

```
require (totalDeposited + wad <= 500_000_000 ether, "Already minted 500M RL1");
```

LAM-33 Dead Code

Informational | Acknowledged

Update

The issue has been acknowledged by the client.
The client provided the following explanation:

```
Not addressed, optimization only
```

File(s) affected: `contracts/Vote.sol`, `contracts/RL1.sol`, `contracts/LL1.sol`

Description: "Dead" code refers to code that is never executed and hence makes no impact on the final result of running a program. Below is an example of dead code in the codebase:

1. In the function `unlockable_10k()`, the inner `if` block cannot be reached in the following snippet:

```
if (filledPeriods >= currentPeriod() + 1) {
    if (currentPeriod() > cumulativeUnlockRates.length) {
        operationX();
    }
}
```

Since we have the property `cumulativeUnlockRates.length == unlockRates.length`, we can replace the blocks with simpler variables, we have:

```
if (A >= B + 1) {
    if (B > A) {
        operationX();
    }
}
```

Since `A >= B + 1 <=> A > B`, we can conclude that we cannot enter in the inner `if` block and execute `operationX()` since we cannot have `A > B && B > A`.

2. L417-418 in the `_update()` function. As the account has to receive funds somehow, an entry in `voteBalanceChanges[from]` will be created at some point before the call. Hence, this condition will never be executed.

Recommendation: We recommend that the team verify if this dead code is needed or if that code is unexpectedly unreachable.

LAM-34 Commented-out Code

Informational | Fixed

Update

The issue has been fixed by the client.
Addressed in: `0381c1eeb991125dd124de76b3ac4590b8132a00`, `e60a4376aebe9b2e376969ada15760639a319192`.
The client provided the following explanation:

```
Fixed in large commit, relevant lines:
  https://github.com/lamina1/rewards/commit/0381c1eeb991125dd124de76b3ac4590b8132a00#diff-
08e835fb020057e51897bc2b9b4de5e6d15a821ec941b1f578262f9840e358daL313
```

File(s) affected: `contracts/RL1.sol`, `contracts/ClaimableAirdrop.sol`

Description: The smart contract contains commented-out code. It is unclear whether this is intentional or not. Furthermore, it makes maintenance and verification of the code cumbersome.

1. L313-320 in the `claim()` function.
2. `ClaimableAirdrop.removeLink()` contains two comments,
    - `//return true; // Return true as the ID was found and removed`
    - `//return false; // Return false if the ID was not found`

Recommendation: We recommend removing or acting on the commented-out code from the contract.

LAM-35

`getLinkedRewards()` Returns Both Claimed and Unclaimed Rewards

Informational | Acknowledged

> **Update**
> The issue has been acknowledged by the client.
> The client provided the following explanation:
>
> ```
> Expected behavior
> ```

File(s) affected: `contracts/ClaimableAirdrop.sol`

Description: `getLinkedRewards()` returns the linked rewards for a recipient address, but it includes both claimed and unclaimed rewards. `airdrop.status.locked` also does not need to be checked as locked airdrops are removed from `activeAirdrops` array.

Recommendation: Consider if this is the expected behavior. If not, update the code accordingly.

LAM-36 Risks Related to Using a Merkle Tree

Informational | Acknowledged

> **Update**
> The issue has been acknowledged by the client.
> The client provided the following explanation:
>
> ```
> Expected behavior
> ```

File(s) affected: `contracts/ClaimableAirdrop.sol`

Description: For each recorded airdrop, the system uses a Merkle Tree to store the details of all eligible rewards. It helps to save gas because only the root of this data structure is stored on-chain. Using such a data structure has associated risks and the following measures can help mitigate these risks:

1. avoid losing the data to build the tree, because it will make it impossible to claim rewards since users will not be able anymore to get the correct data to use for the `proof` parameter;
2. data should be public so any public observer can verify the content of the data structure matching the `merkleRoot` stored on-chain;
3. integrity data should be checked for the leaves of the tree. In particular, no leaf should appear twice in the same tree.

Recommendation: Consider making sure that the measures described above are implemented.

LAM-37

An Exact and Exhaustive List of Operations Allowed per Airdrop State Could Be Done to Make the System More Auditable

Informational | Fixed

> **Update**
> The issue has been fixed by the client.
> Addressed in: `2f0fc0e10c754aa4a9da81165fb3902f4c6d4ab6` .

File(s) affected: `contracts/ClaimableAirdrop.sol`

Description: An airdrop can have different states:

1. paused or not;
2. initialized or not;
3. expired or not;
4. locked or not;

Different operations can be done, but only in some allowed states. However, no exhaustive specification describes for each operation the list of states where it should or should not revert. As a result, unexpected edge cases of the state may be reached.

For example, it is unclear what actions should be allowed in each of these states. See these functions:
1. `linkAddress()` does not check if airdrop exists
2. `revokeRewards()` does not check if airdrop is paused

Recommendation: Consider creating a matrix defining for each operation and airdrop state if the system should revert or not.

LAM-38 Ownership Can Be Renounced

Informational | Acknowledged

Update
The issue has been acknowledged by the client.
The client provided the following explanation:

```
Accepted risk
```

File(s) affected: `contracts/ClaimableAirdrop.sol`

Description: If the owner renounces their ownership, all ownable contracts will be left without an owner. Consequently, any function guarded by the `onlyOwner` modifier will no longer be able to be executed.

Recommendation: Confirm that this is the intended behavior. If not, override and disable the `renounceOwnership()` function in the affected contracts. For extra security, consider using a two-step process when transferring the ownership of the contract (e.g. `Ownable2Step` from OpenZeppelin).

LAM-39
Usage of Distinct Reentrancy Guards Within the Same Contract

Undetermined | Fixed

Update
The issue has been fixed by the client.
Addressed in: `0381c1eeb991125dd124de76b3ac4590b8132a00`.
The client provided the following explanation:

```
Fixed in large commit, relevant lines:
  https://github.com/lamina1/rewards/commit/0381c1eeb991125dd124de76b3ac4590b8132a00#diff-
08e835fb020057e51897bc2b9b4de5e6d15a821ec941b1f578262f9840e358daL569
```

File(s) affected: `contracts/RL1.sol`

Description: Two distinct reentrancy guards are used in the contract `RL1`:

- `nonReentrant` by the functions `_transferFrom()`, `trustedTransfer()` and `_withdrawFor()`;
- `transferGuard` by the function `transferFrom()`;

As a result, any function from one set can be reentered by any function of the other set. This design choice should be clarified.

Recommendation: We recommend clarifying this design choice to confirm that the system is working as expected.

LAM-40
Updates of Unlocking and Mining Rates Can Be Done without the Existence of a Contract `omfmaRules`

Undetermined | Acknowledged

Update
The issue has been acknowledged by the client.
The client provided the following explanation:

```
Accepted risk
```

File(s) affected: `contracts/LL1.sol`, `contracts/RL1.sol`

Description: In the contracts `LL1` and `RL1`, the OMFMA can update the unlock and mining rates via the functions `setUnlockRate()` and `setMiningRate()`. The new value is checked with a call to the contract `omfmaRules`. However, if no `omfmaRules` address is defined, the new value will not be checked and still proceed to the update.

Recommendation: Consider assessing if it is the expected behavior. If not, replace the line `if (address(omfmaRules) != address(0))` with a `require` statement.

LAM-41

Unclear Behavior of the Functions `Vote.findRL1DepositsAsOf()` and `Vote.findVoteBalanceInfoAsOf()` Should Be Clarified

- Undetermined
- Fixed

Update

The issue has been fixed by the client.
Addressed in: `8127b30df2295aec73e0e16500089784cb96661e`.

The client added the comment: We addressed that issue by refactoring the vote contract. The RL1 deposits are now tracked in a map, meaning that there is no more need for the broken findRL1Deposit function. The findVoteBalance function is working as expected, since it has to lookup the most recent vote balance of the user. I've added a "fast path" to this function that checks the last element of the array before performing the binary search. This is because for most users (stake and forget), the last element of the array will be the one to look for when computing rewards for any day.

File(s) affected: `contracts/Rewards.sol`

Description: The function `Vote.findRL1DepositsAsOf()` is used to check when the last `RL1` deposit happened for a given day passed as a parameter.

Using the following code which is a simplified version where `item[i].day = item[i].balance = item[i]`:

```
pragma solidity 0.8.17;

contract Parent{

    uint256[] rl1Deposits = [0,3,5,10];

    function findRL1DepositsAsOf(uint16 _day) public view returns (uint256) {
        if (rl1Deposits.length == 0) {
            return 0;
        }
        if (rl1Deposits[0] > _day) {
            return 0;
        }

        uint256 left = 0;
        uint256 right = rl1Deposits.length - 1;
        uint256 mid = 0;

        while (left < right) {
            mid = left + (right - left) / 2;
            if (rl1Deposits[mid] > _day) {
                right = mid;
            } else {
                left = mid + 1;
            }
        }

        if (rl1Deposits.length > left && rl1Deposits[left] == _day) {
            return rl1Deposits[left];
        }
        if (left > 0 && rl1Deposits[left - 1] <= _day) {
            return rl1Deposits[left - 1];
        }
        return 0;
    }
}
```

We obtain:

- `0` for the values `(0,1,2)`;

- `3` for the values `(3,4)`;
- `5` for the values `(5,6,7,8,9,11,12)`;
- `10` for the values `(10)`;

First, the fact that we obtain `5` for `(11,12)` is unclear and should be clarified by the team to be working as expected or to be a bug.

Then, the fact that we obtain `3` for `3,4` means that `claimableRL1()` will be considered for the days `3` and `4`. This will provide an incorrect value of `6` when the function `claimableRL1Range()` is executed on the range `3,4`. The direct consequence is that, if there are `100` tokens available and `Bob` can claim `50` tokens on day `3`, he cannot call twice `claimRL1(3)` due to the fact that `claimed[Bob][3] = true`, but can claim an additional amount of `50` tokens by calling `claimRL1(4)`. The fact that this second scenario is working as expected or is a bug should also be clarified.

Finally, note that the function `Vote.findVoteBalanceInfoAsOf()` also returns the item before the last one because the same algorithm is used by both functions. This behavior should also be double-checked.

Recommendation: Consider double-checking whether the functions are working as expected or not. If not, adapt the code accordingly and ensure that it does now work as expected with robust tests.

LAM-42 Unclear incentives to vote

• Undetermined Acknowledged

Update

The issue has been acknowledged by the client.
The client provided the following explanation:

```
Accepted risk
```

File(s) affected: `contracts/Vote.sol`, `contracts/Rewards.sol`

Description: For the correct function of the system, the user should lock their `L1` tokens and mint vote tokens to participate in the voting system. The current incentive for the user to vote is not clear, as a user can receive a proportion of `RL1` tokens via `claimRL1()`, but that requires the user to hold the vote tokens and not use them as if they use them the allocation of `RL1` tokens goes down. Hence, the profit from voting and potentially getting selected to win the larger distribution should be more significant than just hoarding the vote tokens.

Getting proper estimations would require game theory modelling.

Recommendation: Consider conducting such simulations to see if there are enough incentives for the users to vote.

LAM-43 The Mechanism of Drips and Subsidy Is Unclear

• Undetermined Acknowledged

Update

The issue has been acknowledged by the client.
The client provided the following explanation:

```
Accepted risk
```

File(s) affected: `contracts/ClaimableAirdrop.sol`

Description: Drips are described as an amount of subsidy drips per recipient. A subsidy is defined as a number to be distributed per airdrop reward to cover gas. A given address can be linked to a maximum of `10` different rewards per airdrop. When the owner links a reward for a node to a recipient address via `linkReward()`, the linked address will receive a drip to claim that reward. However, the amount of drips sent is recorded at the level of the nodes. As a result, it is possible for an address to receive more than `drips` drips. Hence it is possible for one address to receive all of the `totalSubsidy` for the airdrop by linking and unlinking the same address for all of the nodes in the tree.

Also, when an address is linked to a node via `linkAddress()`, that address will receive the value `drip = airdrop.status.subsidy / airdrop.status.drips`. However, when an address claims a reward, it will receive `drip = airdrop.status.subsidy - airdrop.rewardStatus[node].subsidyReceived`, which is different and the fact that this address already received a drip for other nodes of the same airdrop will not be considered.

Recommendation: Consider storing the number of drips received per address per airdrop to make sure that an address cannot receive more than `drips` drips from the subsidy of a given airdrop.

LAM-44 Specific Aspects of the Airdrop Claiming Mechanism Should Be Clarified

• Undetermined Acknowledged

Update

The issue has been acknowledged by the client.
The client provided the following explanation:

```
Accepted risk
```

File(s) affected: `contracts/ClaimableAirdrop.sol`

Description: Nodes in the Merkle Tree corresponding to rewards are identified by a keccak256 hash calculated based on a recipient hash and an amount. The reward for a given node can be claimed by the recipient or on behalf of the recipient by the `owner`. An address can be the recipient of a reward by being linked to that address by the `owner` (via `linkAddress()`) or by having its address match the recipient hash, and by having that node inside a Merkle Tree that has his Merkle root stored. Note that a given node can only be claimed once.

Let's consider the scenario where node `N` = `hash(address of Bob, 100)` is inside the Merkle Tree with a Merkle root stored for airdrop `A`.

There are three main cases:

- Case 1: `owner` linked `N` to Bob for `A` via `linkAddress()`;
- Case 2: `owner` linked `N` to Alice for `A` via `linkAddress()`;
- Case 3: `owner` did not link `N` to an address;

Here are now four scenarios, and their outcome:

1. Bob claims `N` via `claimMyRewards()`: a. for case 1, Bob will get `100` tokens and the link will be removed because `linkedClaim` is `true`; b. for case 2, Bob will get `100` tokens instead of Alice, but the link will not be removed because `linkedClaim` is `false`; c. for case 3, Bob will get `100` tokens, there is no link to remove;
2. Alice claims `N` via `claimMyRewards()`: a. for case 1, it will revert; b. for case 2, Alice will get `100` tokens and the link will be removed because `linkedClaim` is `true`; c. for case 3, it will revert;
3. `Owner` claims `N` for Bob via `claimRewards()`: a. for case 1, Bob will get `100` tokens but the link will not be removed because `linkedClaim` is `false`; b. for case 2, Alice will get `100` tokens instead of Bob, and the link will be removed because `linkedClaim` is `true`; c. for case 3, it will revert;
4. `Owner` claims `N` for Alice via `claimRewards()`: a. for case 1, Bob will get `100` tokens instead of Alice, and the link will not be removed because `linkedClaim` is `false`; b. for case 2, Alice will get `100` tokens and the link will be removed because `linkedClaim` is `true`; c. for case 3, it will revert;

We highlighted two types of outcomes that should be clarified as working as expected or not:

1. The fact that a linked address will get tokens instead of what is requested by the `owner` (scenarios 1b, 3b, and 4a);
2. The fact that a link between a node and an address is not removed even if that node has been claimed (scenarios 1b, 3a, 4a);

Recommendation: Consider clarifying whether the two types of outcomes highlighted are working as expected or not. If not, consider:

1. defining what should have the priority between a linked address and a call of `claimRewards()` by the `owner`;
2. making sure that a link between a node and an address is removed any time the reward associated with that node is claimed.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Code Documentation

In `LL1.sol`:

1. In `setOMFMA()`, what is represented by the value of `secondsPerPeriod * 12 / 10` should be made explicit.
2. Rename `withdrawnTokens` with `totalWithdrawn`, and `totalWithdrawn` with `totalWithdrawnPerUser`;

In `RL1.sol`:

1. Rename `totalDeposited` with `totalMinted`, and `totalMinted` with `totalMinterPerUser`;

In `Rewards.sol`:

1. Rename `sortedAccountsWithBalances()` with `sortedAccountsWithDetails()`;

Adherence to Best Practices

In `LL1.sol` and `RL1.sol`:

1. Remove `import "hardhat/console.sol";` and `debugSpeedup`.
2. Merge the functions setLL1Sender() and removeLL1Sender() in a single function:

```
function updateLL1Sender(address dst, bool newStatus) public onlyOwner {
    if (LL1Senders[dst] != newStatus) LL1Senders[dst] = newStatus;
}
```

3. Use a parent contract `PeriodManager` containing all functions shared by both functions and dealing with periods.
4. Since we only have `unlockRates.length == cumulativeUnlockRates.length`, we can use the same value for both in `unlockable_10k()`.

In `Vote.sol`:

1. Update the type of `rewards` from `IERC20` to `address`;
2. In `receiveRL1()`, the fact that we use the keyword `storage` makes the following line redundant: `rl1Deposits[rl1Deposits.length - 1] = bi`;
3. In `_update()`, the fact that we use the keyword `storage` makes the three lines updating `voteBalanceChanges` redundant;
4. In `lock()`, the cast `uint(_days)` is redundant;
5. Remove the import of `ERC20` and directly inherit `ERC20Burnable` which already inherits `ERC20`.
6. To save gas, cache the value of:
   - `locker[msg.sender][index]` in a `memory` variable in `unlock()` (except the last line `delete locker[msg.sender][index]`);
   - `rl1Deposits.length` in a local variable in `findRL1DepositsAsOf()`;
   - `lockCount[_addr]` in a local variable in `totalLocked()`;
   - `lockCount[_addr]` in a local variable in `totalClaimable()`;
   - `lockCount[msg.sender]` in a local variable in `lock()`;
   - `lockList[_a].length` in a local variable in `unlockable()`;
   - `lockCount[_addr]` in a local variable in `nextClaimable()`;

In `Rewards.sol`:

1. To save gas, cache the value of:
   - `accountAddrs.length` in a local variable in `sortedAccountsWithBalances()`;
   - `topScorers.length` in a local variable in `distributeRewards()`;
   - `distributable * uint(account.balance) / uint(totalScore)` in a local variable in `distributeRewards()`;

In `LL1.sol`:

1. To save gas, cache the value of:
   - `unlockRates.length` in a local variable in `ensureProperRatePeriods()`;

In `RL1.sol`:

1. Remove the function `ensureProperRatePeriods()` if it is not used.
2. Remove the commented code in `claim()`;
3. Call `_transferFrom(rewardsContract,dst,wad)` from `rewardsTransfer()` and `_transferFrom(msg.sender,dst,wad)` from `trustedTransfer()`;
4. Remove the first field of the event `Deposit` since it is always `address(this)`;
5. Create a dedicated function for the lines:

```
uint64 startOfAcceptableCallTime = period * secondsPerPeriod + periodStart - prePeriodWindowStart;
uint64 endOfAcceptableCallTime = period * secondsPerPeriod + periodStart - prePeriodWindowEnd;
require(block.timestamp >= startOfAcceptableCallTime && block.timestamp <= endOfAcceptableCallTime, "Not
in time window");
```

6. To save gas, cache the value of:
   - `claimableReleaseRate.length` and `currentPeriod()` in local variables in `ensureProperReleaseRate()`;
   - `claimableProofTokens.length` and `currentPeriod()` in local variables in `ensureProperRewardPeriods()`;
   - `rate * balanceOf[address(this)] / 10000 / 100` in a local variable in `setMiningRate()`;
   - `claimableReleaseRate[index] / 10000 / 100` in a local variable in `mintRL1()`;

Adherence to Specification

1. Fixed Lots of TODOs left in the spec

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.10.0

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

All issues identified by slither have either been flagged as false positives or addressed in the report.

Test Suite Results

Main repository

All 83 tests are passing.

Airdrop Contract

All 74 tests are passing.

Fix review update

Main repository

All 83 tests are passing.

Airdrop Contract

All 124 tests are passing.

50 additional tests were included since the start of the audit.

```
**Main repository**

  LL1 Contract
    Deployment
      ✔ Should set the right owner (797ms)
    Unfinished Deployment
      ✔ Should return 0 for unlockable amount
    Transactions
      ✔ Should allow deposit and emit Deposit event
  Unlocking and Locking
    ✔ Should not increment periods after 7883999  seconds
    ✔ Should increment periods only after 7884000 seconds
    ✔ Should unlock 10% of the unlock rate at 788400 seconds through the first period.
    ✔ Should show percent of current period accurately throughout the period.
    ✔ Should do period based catchup if nobody has entered data in a while
  Rate Tests
    ✔ Should not allow rates to be set over 100%
    ✔ Should not return a rate over 100%
    - Should not allow two consecutive zero rate periods, defaulting to 2.5% if the last was at 0%
    - Should not allow rates to move more than 1.5% per period
  Period Tests
    ✔ Should not allow rate setting after the update window
    ✔ Should not allow rate setting before the update window
    ✔ Should allow rate setting in the update window
    ✔ Should fill in missing periods when we've been lazy updating the rate
  Ownership and Permissions
    ✔ Should only allow OMFMA to set unlock rates
    ✔ Should correctly transfer ownership only on ownership
    ✔ Should transfer ownership on acceptance
  Withdrawal
    ✔ Should allow withdrawal calls
    ✔ Should not allow withdrawal if you have never deposited or received a deposit
    ✔ Should calculate withdrawable as 0 if you just withdrew.
```

```
      ✔ Should calculate withdrawable properly if you withdrew a while ago.
      ✔ Should complain if you withdrew then transferred more than you have.
      ✔ Should calculate withdrawable properly if you withdrew, then transferred, then waited and
withdrew .
    Transfer and Approval
      ✔ Should allow transfers within unlock limits
      ✔ Should enforce transfer restrictions based on ownership or allowance
    Bulk Transfers
      ✔ Should reject mismatched amt and recip length
      ✔ Should reject 0 value transfers
      ✔ Should reject bulk transfers to the 0 address
      ✔ Should support bulk transfers (62ms)

  RL1 Contract
    Deployment
      ✔ Should set the right owner
    Unfinished Deployment
      ✔ Should return 0 for unlockable amount
    Transactions
      ✔ Should allow deposit and emit Deposit event
    Unlocking and Locking
      ✔ Should not increment periods after 7883999  seconds
      ✔ Should increment periods only after 7884000 seconds
      ✔ Should unlock 10% of the unlock rate at 788400 seconds through the first period.
Checking for 5 eth claimable this period
      ✔ Should show 5%* 1000 eth/ 10  0.5% of 1000 eth = 5 eth claimable at 788400 seconds through the
first period.
      ✔ Should show percent of current period accurately throughout the period.
      ✔ Should do period based catchup if nobody has entered data in a while
    Rate Tests
      ✔ Should not allow rates to be set over 100%
      ✔ Should not return a rate over 100%
      - Should not allow two consecutive zero rate periods, defaulting to 2.5% if the last was at 0%
      - Should not allow rates to move more than 1.5% per period
    Period Tests
      ✔ Should not allow rate setting after the update window
      ✔ Should not allow rate setting before the update window
      ✔ Should allow rate setting in the update window
      ✔ Should fill in missing periods when we've been lazy updating the rate
    Ownership and Permissions
      ✔ Should only allow OMFMA to set unlock rates
      ✔ Should correctly transfer ownership only on ownership
      ✔ Should transfer ownership on acceptance
    Withdrawal
      ✔ Should allow withdrawal calls
      ✔ Should not allow withdrawal if you have never deposited or received a deposit
      ✔ Should calculate withdrawable as 0 if you just withdrew.
      ✔ Should calculate withdrawable properly if you withdrew a while ago.
      ✔ Should complain if you withdrew then transferred more than you have.
      ✔ Should calculate withdrawable properly if you withdrew, then transferred, then waited and
withdrew .
    Transfer and Approval
      ✔ Should allow transfers within unlock limits
      ✔ Should enforce transfer restrictions based on ownership or allowance
    RL1 Token Movements
      ✔ Should start out with the full number of tokens owned by the contract itself
      ✔ Should send tokens to the rewards contract after claim() is called but not before
      - Should do the compounding / geometric runoff from successive rates properly

  Rewards Contract
    Deployment
      ✔ Should set the right owner (49ms)
    Locking
      ✔ Should allow lock and emit Lock event
      - Should allow unlock after the proper time period, and return L1
      - Should allow locking of first a long duration then a short duration then a long duration then
properly remove the middle short duration on unlock
    Transactions
      ✔ Should allow vote and emit Vote event
      ✔ Should allow distributeRewards (39ms)
      - Should do end to end test of locking, voting, receiving, and relocking
    Ownership and Permissions
```

```
    ✔ Should only allow owner to set voting contract
    ✔ Should allow an arbitrary RL1Sender to call trustedTransfer, but not just anyone (42ms)
    ✔ Should disallow a former RL1Sender calling safeTransfer (44ms)

  Vote Contract
    Deployment
      ✔ Should set the right rewards contract (52ms)
    Locking
      ✔ Should allow locking tokens
      ✔ Should not allow locking for more than 3650 days
      ✔ Should not allow locking with zero amount
    Unlocking
      ✔ Should allow unlocking L1 after lock period
      ✔ Should not allow unlocking before lock period
      ✔ Should allow multiple locks with unlocks in order
      ✔ Should allow multiple locks with unlocks in random order
      ✔ Should provide a list of periods that can be unlocked for a given address
    Claiming RL1
      ✔ Should allow claiming RL1 tokens for a past day (46ms)
      ✔ Should not allow claiming for current or future day
      ✔ Should not allow claiming twice for same day (44ms)
      ✔ Should allow burning of tokens
      ✔ Should not allow transfer of vote tokens at launch
      ✔ Should allow transfer of vote tokens after unpause
      ✔ Should update accounting of Balance Info properly after transfer of VOTE tokens
    Permissions
      ✔ Should only allow owner to set rewards contract
      ✔ Should only allow owner to set start day
      ✔ Should only allow rewards contract to deposit RL1


  83 passing (2s)
  8 pending

**Airdrop Contract**

  ClaimableAirdrop Integration with LL1 Token
Rewards generated
Generated rewards merkle tree with 200 users
Deploying LL1 contract
Minting LL1
Deploying airdrop contract
Initial LL1 balance in Airdrop Contract:   101000000000000000000000
Total LL1 allocated for generated airdrop: 101000000000000000000000
Initial eth balance in Airdrop Contract:   20000000000000000000
Gas used for claiming a single reward: 219657
Testing user claiming Discord rewards
Testing revoking and unrevoking batches of rewards
Gas used for revoking 10 users: 363168
Gas used for unrevoking 10 users: 140616
    ✔ should perform end-to-end integration testing with LL1 token (3771ms)

  ClaimableAirdrop
    Airdrop Creation
      ✔ should initialize an airdrop with valid parameters
      ✔ should create multiple airdrops and increment lastAirdropId
      ✔ should revert when creating an airdrop with expiration in the past
      ✔ should revert when creating an airdrop with insufficient token balance
      ✔ should revert when creating an airdrop by non-owner
    Claiming Rewards
      ✔ should allow address recipients to claim rewards (420ms)
      ✔ should allow Discord recipients to claim rewards (473ms)
      ✔ should allow recipients to claim multiple rewards (461ms)
      ✔ should allow owner to claim multiple rewards (62ms)
      ✔ should update the claimedAmount of the airdrop when claiming a reward (45ms)
      ✔ should revert when claiming a reward from a non-existent airdrop
      ✔ should revert when claiming rewards with an invalid Merkle proof (4222ms)
      ✔ should revert when claiming a reward from an expired airdrop
      ✔ should revert when claiming a reward multiple times for the same recipient (49ms)
      ✔ should revert when claiming a reward from a locked airdrop
      ✔ should revert when claiming a revoked reward
    Locking and Unlocking Airdrops
```

```
    ✔ should lock a revocable airdrop and set the locked flag to true
    ✔ should unlock a locked airdrop and set the locked flag to false
    ✔ should lock an expired nonrevocable airdrop
    ✔ should lock all expired airdrops
    ✔ should lock an unexpired revocable airdrop
    ✔ should update allocation and subsidies correctly when locking an airdrop
    ✔ should update totalAllocatedAmount and totalClaimedAmount correctly when unlocking an airdrop
    ✔ should revert when locking a non-revocable airdrop that has not expired
    ✔ should revert when locking an airdrop that is already locked
    ✔ should revert when unlocking an airdrop that is not locked
    ✔ should revert when unlocking an airdrop that has expired
    ✔ should revert when unlocking an airdrop without sufficient token balance
    ✔ should revert when unlocking an airdrop without sufficient ether balance
  Revoking and Unrevoking Rewards
    ✔ should revoke an address reward and update the revoked mapping
    ✔ should not update the revoked mapping when revoking an already claimed reward
    ✔ should revert when revoking a reward from a non-revocable airdrop
    ✔ should revert when revoking a reward from a locked airdrop
    ✔ should unrevoke a reward and update the revoked mapping
    ✔ should not update the revoked mapping when unrevoking a reward that is not currently revoked
    ✔ should revert when unrevoking a reward from a non-revocable airdrop
    ✔ should revert when unrevoking a reward from a locked airdrop
    ✔ should revert when unrevoking a reward when there is insufficient token balance in the contract
    ✔ should update allocations and subsidies correctly when revoking and unrevoking a reward
  Setting Allocation
    ✔ should set the allocation of an airdrop correctly when increasing
    ✔ should set the allocation of an airdrop correctly when decreasing
    ✔ should set the allocation of a locked airdrop
    ✔ should update totalAllocation correctly when increasing the allocation
    ✔ should update totalAllocatedAmount correctly when decreasing the allocation
    ✔ should revert when setting the allocation with insufficient token balance in the contract
  Linking Discord ID to EVM Address
    ✔ should link a Discord ID to an EVM address
    ✔ should link multiple Discord IDs to different EVM addresses (82ms)
    ✔ should revert when linking a Discord ID to an invalid EVM address
    ✔ should override the linked EVM address when linking a Discord ID that is already linked
  Withdrawing Unallocated Tokens
    ✔ should withdraw unallocated reward tokens to a specified recipient
    ✔ should withdraw unallocated ether to a specified recipient
    ✔ should withdraw other ERC20 tokens accidentally sent to the contract
    ✔ should revert when withdrawing tokens with insufficient unallocated balance
  Pausing and Unpausing the Airdrop
    ✔ should pause an airdrop when a claim exceeds the current allocated amount
    ✔ should pause an airdrop when a subsidy exceeds the current available amount (95ms)
    ✔ should unpause a paused airdrop
    ✔ should revert when claiming or linking rewards from a paused airdrop
    ✔ should revert when unpausing an airdrop that is not paused
  Access Control
    ✔ should revert when executing owner-only functions with non-owner accounts (46ms)
  Edge Cases and Boundary Conditions
    ✔ should lock an airdrop at the exact expiration time
    ✔ should revert when unlocking an airdrop at the exact expiration time
    ✔ should revoke and unrevoke rewards with the maximum possible amounts
    ✔ should create an airdrop with a very large expiration time
    ✔ should claim rewards from an airdrop that has been locked and then unlocked
    ✔ should claim rewards from an airdrop that has been paused and then unpaused
    ✔ should keep revocation valid after extending the airdrop's expiration time
    ✔ should set the allocation of an airdrop to zero and revert when claiming rewards
    ✔ should withdraw reward tokens when the total allocated amount is close to the contract's balance
    ✔ should unlock an airdrop when the contract's balance is insufficient but becomes sufficient
  Getting Data
    ✔ should find all active linked rewards for an address (66ms)
    ✔ should return all active Airdrops
    ✔ should return airdrop without the large mappings
    ✔ should return the linked address of a Discord reward


  74 passing (13s)
```

Fix review update

Main repository

```
LL1 Contract
  Deployment
    ✔ Should set the right owner (705ms)
  Unfinished Deployment
    ✔ Should return 0 for unlockable amount
  Transactions
    ✔ Should allow deposit and emit Deposit event
  Unlocking and Locking
    ✔ Should not increment periods after 7883999  seconds
    ✔ Should increment periods only after 7884000 seconds
    ✔ Should unlock 10% of the unlock rate at 788400 seconds through the first period.
    ✔ Should show percent of current period accurately throughout the period.
    ✔ Should do period based catchup if nobody has entered data in a while
  Rate Tests
    ✔ Should not allow rates to be set over 100%
    ✔ Should not return a rate over 100%
    - Should not allow two consecutive zero rate periods, defaulting to 2.5% if the last was at 0%
    - Should not allow rates to move more than 1.5% per period
  Period Tests
    ✔ Should not allow rate setting after the update window
    ✔ Should not allow rate setting before the update window
    ✔ Should allow rate setting in the update window
    ✔ Should fill in missing periods when we've been lazy updating the rate
  Ownership and Permissions
    ✔ Should only allow OMFMA to set unlock rates
    ✔ Should correctly transfer ownership only on ownership
    ✔ Should transfer ownership on acceptance
  Withdrawal
    ✔ Should allow withdrawal calls
    ✔ Should not allow withdrawal if you have never deposited or received a deposit
    ✔ Should calculate withdrawable as 0 if you just withdrew.
    ✔ Should calculate withdrawable properly if you withdrew a while ago.
    ✔ Should complain if you withdrew then transferred more than you have.
    ✔ Should calculate withdrawable properly if you withdrew, then transferred, then waited and
withdrew .
  Transfer and Approval
    ✔ Should allow transfers within unlock limits
    ✔ Should enforce transfer restrictions based on ownership or allowance
  Bulk Transfers
    ✔ Should reject mismatched amt and recip length
    ✔ Should reject 0 value transfers
    ✔ Should reject bulk transfers to the 0 address
    ✔ Should support bulk transfers

RL1 Contract
  Deployment
    ✔ Should set the right owner
  Unfinished Deployment
    ✔ Should return 0 for unlockable amount
  Transactions
    ✔ Should allow deposit and emit Deposit event
  Unlocking and Locking
    ✔ Should not increment periods after 7883999  seconds
    ✔ Should increment periods only after 7884000 seconds
    ✔ Should unlock 10% of the unlock rate at 788400 seconds through the first period.
Checking for 5 eth claimable this period
    ✔ Should show 5%* 1000 eth/ 10  0.5% of 1000 eth = 5 eth claimable at 788400 seconds through the
first period.
    ✔ Should show percent of current period accurately throughout the period.
    ✔ Should do period based catchup if nobody has entered data in a while
  Rate Tests
    ✔ Should not allow rates to be set over 100%
    ✔ Should not return a rate over 100%
    - Should not allow two consecutive zero rate periods, defaulting to 2.5% if the last was at 0%
    - Should not allow rates to move more than 1.5% per period
  Period Tests
    ✔ Should not allow rate setting after the update window
    ✔ Should not allow rate setting before the update window
    ✔ Should allow rate setting in the update window
    ✔ Should fill in missing periods when we've been lazy updating the rate
  Ownership and Permissions
```

```
      ✔ Should only allow OMFMA to set unlock rates
      ✔ Should correctly transfer ownership only on ownership
      ✔ Should transfer ownership on acceptance
    Withdrawal
      ✔ Should allow withdrawal calls
      ✔ Should not allow withdrawal if you have never deposited or received a deposit
      ✔ Should calculate withdrawable as 0 if you just withdrew.
      ✔ Should calculate withdrawable properly if you withdrew a while ago.
      ✔ Should complain if you withdrew then transferred more than you have.
      ✔ Should calculate withdrawable properly if you withdrew, then transferred, then waited and
withdrew .
    Transfer and Approval
      ✔ Should allow transfers within unlock limits
      ✔ Should enforce transfer restrictions based on ownership or allowance
    RL1 Token Movements
      ✔ Should start out with the full number of tokens owned by the contract itself
      ✔ Should send tokens to the rewards contract after claim() is called but not before
      - Should do the compounding / geometric runoff from successive rates properly

  Rewards Contract
    Deployment
      ✔ Should set the right owner (40ms)
    Locking
      ✔ Should allow lock and emit Lock event
      - Should allow unlock after the proper time period, and return L1
      - Should allow locking of first a long duration then a short duration then a long duration then
properly remove the middle short duration on unlock
    Transactions
      ✔ Should allow vote and emit Vote event
      ✔ Should allow distributeRewards
      - Should do end to end test of locking, voting, receiving, and relocking
    Ownership and Permissions
      ✔ Should only allow owner to set voting contract
      ✔ Should allow an arbitrary RL1Sender to call trustedTransfer, but not just anyone
      ✔ Should disallow a former RL1Sender calling safeTransfer

  Vote Contract
    Deployment
      ✔ Should set the right rewards contract
    Locking
      ✔ Should allow locking tokens
      ✔ Should not allow locking for more than 3650 days
      ✔ Should not allow locking with zero amount
    Unlocking
      ✔ Should allow unlocking L1 after lock period
      ✔ Should not allow unlocking before lock period
      ✔ Should allow multiple locks with unlocks in order
      ✔ Should allow multiple locks with unlocks in random order
      ✔ Should provide a list of periods that can be unlocked for a given address
    Claiming RL1
      ✔ Should allow claiming RL1 tokens for a past day
      ✔ Should not allow claiming for current or future day
      ✔ Should not allow claiming twice for same day
      ✔ Should allow burning of tokens
      ✔ Should not allow transfer of vote tokens at launch
      ✔ Should allow transfer of vote tokens after unpause
      ✔ Should update accounting of Balance Info properly after transfer of VOTE tokens
    Permissions
      ✔ Should only allow owner to set rewards contract
      ✔ Should only allow owner to set start day
      ✔ Should only allow rewards contract to deposit RL1


  83 passing (1s)
  8 pending


**Airdrop Contract**

  ClaimableAirdrop Integration with LL1 Token
Rewards generated
Generated rewards merkle tree with 200 users
```

```
Deploying LL1 contract
Minting LL1
Deploying airdrop contract
Initial LL1 balance in Airdrop Contract:   10100000000000000000000
Total LL1 allocated for generated airdrop: 10100000000000000000000
Initial eth balance in Airdrop Contract:   200000000000000000000
Gas used for claiming a single reward: 214621
Testing user claiming Discord rewards
Testing revoking and unrevoking batches of rewards
Gas used for revoking 10 users: 437590
Gas used for unrevoking 10 users: 141842
    ✔ should perform end-to-end integration testing with LL1 token (29472ms)

  ClaimableAirdrop
    Airdrop Creation
      ✔ should initialize an airdrop with valid parameters (134ms)
      ✔ should create multiple airdrops and increment lastAirdropId (96ms)
      ✔ should revert when creating too many airdrops (517ms)
      ✔ should revert when creating an airdrop with expiration in the past
      ✔ should revert when unlocking a new airdrop with insufficient token balance (54ms)
      ✔ should revert when unlocking a new airdrop with insufficient ether balance (70ms)
      ✔ should revert when creating an airdrop by non-owner
      ✔ should revert when creating an airdrop with no allocation
      ✔ should revert when creating an airdrop with no rewards
      ✔ should rever when using invalid subsidy params (64ms)
    Claiming Rewards
      ✔ should allow address recipients to claim rewards (2240ms)
      ✔ should allow Discord recipients to claim rewards (1268ms)
      ✔ should allow recipients to claim multiple rewards (669ms)
      ✔ should update the claimedAmount of the airdrop when claiming a reward (167ms)
      ✔ should revert when claiming a reward from a non-existent airdrop (133ms)
      ✔ should revert when claiming rewards with an invalid Merkle proof (34475ms)
      ✔ should revert when claiming rewards with unequal array length input (252ms)
      ✔ should revert when claiming a reward from an expired airdrop (152ms)
      ✔ should revert when claiming a reward multiple times for the same recipient (249ms)
      ✔ should revert when claiming a reward from a locked airdrop (97ms)
      ✔ should revert when claiming a revoked reward (202ms)
      ✔ should revert claimMyRewards when the message sender is not the recipient (84ms)
      ✔ should allow any privileged role to use claimRewards (316ms)
    Locking and Unlocking Airdrops
      ✔ should lock a revocable airdrop and set the locked flag to true
      ✔ should unlock a locked airdrop and set the locked flag to false (39ms)
      ✔ should lock an expired nonrevocable airdrop (53ms)
      ✔ should lock all expired airdrops (150ms)
      ✔ should lock an unexpired revocable airdrop (52ms)
      ✔ should update allocation and subsidies correctly when locking an airdrop (202ms)
      ✔ should update totalAllocatedAmount and totalClaimedAmount correctly when unlocking an airdrop
(196ms)
      ✔ should revert when locking a non-revocable airdrop that has not expired
      ✔ should revert when locking an airdrop that is already locked
      ✔ should revert when locking an airdrop that hasn't been created
      ✔ should revert when unlocking an airdrop that hasn't been created
      ✔ should revert when unlocking an airdrop that is not locked
      ✔ should revert when unlocking an airdrop that has expired
      ✔ should revert when unlocking an airdrop without sufficient token balance (61ms)
      ✔ should revert when unlocking an airdrop without sufficient ether balance (52ms)
      ✔ should revert when locking, unlocking, or extending airdrops from non-admin roles (204ms)
      ✔ should revert when extending an airdorp with invalid parameters (65ms)
      ✔ should allow managers to perform lock/unlock/extend (72ms)
    Revoking and Unrevoking Rewards
      ✔ should revoke an address reward and update the revoked mapping
      ✔ should not update the revoked mapping when revoking an already claimed reward (504ms)
      ✔ should skip revoking rewards if they've already been revoked (59ms)
      ✔ should revert if any of the proofs are invalid (89ms)
      ✔ should revert when revoking a reward from an unrevocable airdrop (82ms)
      ✔ should revert when revoking a reward from a locked airdrop (41ms)
      ✔ should unrevoke a reward and update the revoked mapping (52ms)
      ✔ should not update the revoked mapping when unrevoking a reward that is not currently revoked
      ✔ should revert when unrevoking a reward from a non-revocable airdrop (80ms)
      ✔ should revert when unrevoking a reward from a locked airdrop (55ms)
      ✔ should revert when unrevoking a reward when there is insufficient token balance in the contract
(79ms)
```

```
    ✔ should revert when revoking a reward with invalid parameters (55ms)
    ✔ should revert when unrevoking a reward with invalid parameters (115ms)
    ✔ should update allocations and subsidies correctly when revoking and unrevoking a reward (299ms)
    ✔ should revert when using a non admin account (124ms)
    ✔ should allow managers to revoke/unrevoke (263ms)
  Setting Allocation
    ✔ should set the allocation of an airdrop correctly when increasing (39ms)
    ✔ should set the allocation of an airdrop correctly when decreasing
    ✔ should set the allocation of a locked or paused airdrop (204ms)
    ✔ should update totalAllocation correctly when increasing the allocation (54ms)
    ✔ should update totalAllocatedAmount correctly when decreasing the allocation (45ms)
    ✔ should revert when setting the allocation with invalid parameters (72ms)
    ✔ should revert when setting the allocation with insufficient token balance in the contract (54ms)
    ✔ should revert when using a non-admin account (63ms)
    ✔ should allow a manager to set the allocation
  Setting number of rewards
    ✔ should set the number of rewards correctly when increasing (53ms)
    ✔ should set the number of rewards correctly when decreasing
    ✔ should set the number of rewards of a locked or paused airdrop (204ms)
    ✔ should update totalSubsidy correctly when increasing the number of rewards (51ms)
    ✔ should update totalSubsidy correctly when decreasing the number of rewards (41ms)
    ✔ should revert when setting the numRewards with insufficient ether balance in the contract (54ms)
    ✔ should revert when setting the numRewards with invalid parameters (56ms)
    ✔ should revert when using a non-admin account (65ms)
    ✔ should allow a manager to set the number of rewards
  Linking Discord ID to EVM Address
    ✔ should link a Discord ID to an EVM address (95ms)
    ✔ should link multiple Discord IDs to different EVM addresses (594ms)
    ✔ should unlink a node after claiming when multiple nodes are linked (186ms)
    ✔ should revert when linking a Discord ID to an invalid EVM address
    ✔ should revert when linking to an already claimed node (140ms)
    ✔ should revert when linking to a revoked reward (39ms)
    ✔ should override the linked EVM address when linking a Discord ID that is already linked (54ms)
    ✔ should allow all roles to use the link command (57ms)
    ✔ should revert when trying to link more than 10 nodes to an address (173ms)
  Withdrawing Unallocated Tokens
    ✔ should withdraw unallocated reward tokens to a specified recipient (77ms)
    ✔ should withdraw unallocated ether to a specified recipient (158ms)
    ✔ should withdraw other ERC20 tokens accidentally sent to the contract (94ms)

    ✔ should allow managers but not bots to withdraw unallocated tokens and ether to an owner (349ms)
    ✔ should revert when calling withdrawTokens on the Reward Token (80ms)
    ✔ should revert when withdrawing tokens with insufficient unallocated balance (88ms)
    ✔ should revert with invalid ether amount (69ms)
  Pausing and Unpausing the Airdrop
    ✔ should pause an airdrop when a claim exceeds the current allocated amount (117ms)
    ✔ should pause an airdrop when a subsidy exceeds the current available ether (398ms)
    ✔ should pause an airdrop when a subsidy exceeds the remaining in an airdrop (210ms)
    ✔ should unpause a paused airdrop (134ms)
    ✔ should revert when claiming or linking rewards from a paused airdrop (166ms)
    ✔ should revert when unpausing an airdrop that is not paused
    ✔ should revert when a non-admin tries to unpause (54ms)
    ✔ should allow a manager to unpause the airdrop (131ms)
  Access Control
    Owner-only functions
      ✔ should allow owner to execute owner-only functions (481ms)
      ✔ should revert when non-owner executes owner-only functions (210ms)
      ✔ should revert when adding more than 10 owners (130ms)
      ✔ should revert when adding more than 10 managers (127ms)
      ✔ should revert when adding more than 10 bots (132ms)
    Manager-only functions
      ✔ should allow manager to execute manager-only functions (175ms)
      ✔ should revert when non-manager executes manager-only functions (71ms)
    Access by other roles
      ✔ should allow admin roles (owner, manager, bot) to execute common functions (100ms)
      ✔ should revert when unauthorized roles attempt common functions
    Closing contract
      ✔ Should restrict removing the last owner (139ms)
  Edge Cases and Boundary Conditions
    ✔ should lock an airdrop at the exact expiration time (76ms)
    ✔ should revert when unlocking an airdrop at the exact expiration time (81ms)
    ✔ should revoke and unrevoke rewards with the maximum possible amounts (342ms)
```

```
      ✔ should create an airdrop with a very large expiration time (53ms)
      ✔ should claim rewards from an airdrop that has been locked and then unlocked (191ms)
      ✔ should claim rewards from an airdrop that has been paused and then unpaused (195ms)
      ✔ should keep revocation valid after extending the airdrop's expiration time (121ms)
      ✔ should withdraw reward tokens when the total allocated amount is close to the contract's balance
(93ms)
      ✔ should unlock an airdrop when the contract's balance is insufficient but becomes sufficient
(130ms)
    Getting Data
      ✔ should find all active linked rewards for an address (523ms)
      ✔ should return all active Airdrops
      ✔ should return airdrop without the large mappings
      ✔ should return the linked address of a Discord reward
      ✔ should revert when getting an uninitialized airdrop (93ms)


  124 passing (2m)
```

Code Coverage

The code coverage is low. We recommend the team improve their test suite to achieve at least 90% branch coverage, and ensure all integrations are properly tested.

Fix review update

The code coverage was improved for almost all files, except for the Vote contract.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
LL1.sol	76.85	52.68	73.33	79.17	... 459,460,462
RL1.sol	82.55	58.09	87.5	86.36	... 586,587,626
Rewards.sol	76.74	47.5	63.64	79.66	... 127,184,193
Vote.sol	74.11	69.77	76	74.48	... 342,418,430
ClaimableAirdrop.sol	94.27	72.48	96.43	94.68	... 496,498,639

Fix review update

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
LL1.sol	87.76	63.79	85.19	85.4	... 426,427,429
RL1.sol	91.45	64.58	90	91.75	... 561,586,587
Rewards.sol	78.57	41.67	40	78.08	... 146,205,214
Vote.sol	37.8	40.54	42.86	41.67	... 351,366,376
ClaimableAirdrop.sol	99.08	89.29	94.74	98.97	736,749,753

Changelog

- 2024-04-25 - Initial report

- 2024-06-10 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT