Lisk Governance, Staking, Reward, Vesting, and Airdrop Contracts

Security Assessment

May 28, 2024

Prepared for:
Jan Hackfeld
Lisk

Prepared by: **Elvis Skoždopolj, Priyanka Bose, and Maciej Domanski**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Sam Greenup, Project Manager
sam.greenup@trailofbits.com

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultants were associated with this project:

Elvis Skoždopolj, Consultant
elvis.skozdopolj@trailofbits.com

Priyanka Bose, Consultant
priyanka.bose@trailofbits.com

Maciej Domanski, Consultant
maciej.domanski@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
April 19, 2024	Pre-project kickoff call
April 26, 2024	Status update meeting #1
May 6, 2024	Delivery of report draft
May 6, 2024	Report readout meeting
May 28, 2024	Delivery of comprehensive report

Executive Summary

Engagement Overview

Lisk engaged Trail of Bits to review the security of its governance, staking, reward, vesting, and airdrop smart contracts, as well as its airdrop Merkle tree generation and back-end components. For users to participate in the governance system, they first need to lock some amount of LSK tokens in the `L2Staking` contract to mint voting power tokens. They can do this through the `L2Staking` contract directly or by using the `L2Reward` contract to earn rewards in LSK tokens on their staked positions. The `L2VestingWallet` and `L2Airdrop` smart contracts are stand-alone components; the `L2VestingWallet` contract gradually unlocks users' vested token amounts over time, and the `L2Airdrop` contract allows users to claim token airdrops by submitting Merkle proofs and completing different tasks to unlock portions of their airdrops.

The `claim-backend` component feeds details of user claims (stored in Merkle tree leaf nodes) to the front end; these details are provided by its endpoint. The component also stores partial signatures on claims from multisignature addresses. The `tree-builder` component builds a Merkle tree from a snapshot of all Lisk users and their balances and computes the Merkle root.

A team of two consultants conducted the review from April 22 to May 3, 2024, for a total of four engineer-weeks of effort. Our testing efforts focused on determining whether it is possible to manipulate the system in order to mint a large amount of voting power tokens, whether governance proposals can be front-run, whether users can bypass the airdrop limits or Merkle tree verification, whether users with locked positions can bypass the amount and duration limits or reduce their penalties, and whether expired positions are correctly accounted for. We checked whether users can influence or corrupt the internal accounting of the `L2Reward` contract to gain more rewards, reduce other users' rewards, or cause the system to enter an invalid state. For the off-chain components, we focused on finding vulnerabilities to major potential threats (like injections) and assessing the general correctness of parsing data.

With full access to source code and documentation, we performed static and dynamic testing of the targets, using automated and manual processes.

Observations and Impact

The Lisk team has taken great care to ensure that users are unable to bypass the expected user flows in order to exploit the system, through robust access controls and data validation that is performed in each function and component. However, the state sync between the different components and the validation of rules in the context of combinations of user actions could be improved. For example, due to a flaw in the `increaseLockingAmount` function's data validation, users could take a series of actions

that will allow them to bypass the minimum duration for staked positions (TOB-LSK2-1). Additionally, the accounting of expired positions is incorrect (TOB-LSK2-9), which could result in lower user rewards, allow users to bypass the rewards limit, or even cause user assets to be frozen. Some code duplication and redundant validation is present in the codebase, which increases the complexity of the system; however, this can be easily resolved by adding such logic to a helper contract or library and replacing all duplicated code with calls to that contract or library.

The attack surface of the `tree-builder` and `claim-backend` components is minimal because users do not usually interact with most of the components' functions; however, we did identify several low-severity issues impacting these components. We also found one medium-severity issue in the `claim-backend` component's `DB` class (TOB-LSK2-3): the server certificate validation is completely disabled even if the environment variable that sets TLS is set to `true`. These issues indicate that Lisk's secure coding practices when developing off-chain components could be improved.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that Lisk take the following steps prior to deploying the contracts:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.
- **Improve the testing suite.** We identified some significant gaps in the testing coverage, especially concerning the `L2Reward` contract's global state accounting. Write additional tests that cover each function's expected side effects and assertions, and consider creating fuzz tests with Echidna, Medusa, or Foundry to test the global accounting of the `L2Reward` contract. See appendix D for information regarding notable functions and features that lack sufficient tests.
- **Ensure that the `claim-backend` component accompanying the front end is ready for production.** This includes setting the `production` flag in Express (TOB-LSK2-7), configuring HTTP security headers on the front end, and ensuring that no services (such as the database) are unnecessarily exposed to the internet. Also, check that the web application is resistant against potential denial-of-service attacks (from both potentially malicious input in parameters and excessive use of the API).

Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	Count
High	0
Medium	2
Low	3
Informational	7
Undetermined	0

CATEGORY BREAKDOWN

Category	Count
Auditing and Logging	1
Configuration	1
Cryptography	1
Data Exposure	2
Data Validation	4
Denial of Service	2
Patching	1

Project Goals

The engagement was scoped to provide a security assessment of the Lisk smart contracts and off-chain components. Specifically, we sought to answer the following non-exhaustive list of questions:

- Are access controls sufficient to prevent users from manipulating the system?
- Can a user mint more voting power tokens than they are entitled to through a combination of actions?
- Are expired positions correctly accounted for?
- Is voting power correctly adjusted in all cases?
- Does the system use any vulnerable dependencies?
- Does the system correctly integrate external dependencies?
- Can the Merkle tree verification be bypassed?
- Can an intermediate node of the Merkle tree be verified as a leaf node?
- Can the airdrop tasks be bypassed?
- Are bitwise operations performed correctly?
- Can users manipulate the internal accounting of the `L2Reward` contract?
- Is the governance system vulnerable to front-running?

Project Targets

The engagement involved a review and testing of the targets listed below.

`L2Staking`, `L2Governor`, `L2VestingWallet`, and `L2Reward` contracts

Repository	https://github.com/LiskHQ/lisk-contracts
Version	`7e683b37f4bcbd5157965b9d2fefc9980acf3bde`
Type	Solidity
Platform	Ethereum

`L2Airdrop` contract

Repository	https://github.com/LiskHQ/lisk-contracts
Version	`57b83735e6057ce0f94dfc28a8cf736eba100faa`
Type	Solidity
Platform	Ethereum

`tree-builder` and `claim-backend`

Repository	https://github.com/LiskHQ/lisk-token-claim
Version	`4ca9d9a72d5da2db66cec9e2888fd8089589c532`
Type	TypeScript
Platform	Node.js

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- **Staking and governance contracts**: The `L2Staking`, `L2Reward`, and `L2VotingPower` contracts manage the staking, locking, and governance functionality within the L2 network. They allow for the creation, modification, and deletion of locked positions, as well as the ability to query such positions based on owner. Their responsibilities also include the minting and burning of NFTs for each locking position. Furthermore, the `L2Staking` and `L2Reward` contracts interact with the `L2VotingPower` contract to adjust the voting power of owners of locked positions.

  During the review process, we looked for issues related to access control violations, issues that could cause users or the protocol to lose funds, unintended ways that users could increase their voting power or modify their positions, and lack of sufficient event generations after critical operations. We found issues related to insufficient event generation (TOB-LSK2-10), a way for users to bypass the minimum lock duration (TOB-LSK2-1), and an erroneous penalty calculation in the fast unlocking mechanism (TOB-LSK2-11).

  - **`L2Reward` contract**: This contract manages L2 staking rewards, the creation, deletion, and modification of user staked positions, and the computation of rewards owed to users based on the token amount and duration of their positions.

    We evaluated the `L2Reward` contract to look for issues associated with inaccurate reward computations, incorrect updates to critical storage variables, ways that users could claim more rewards than they are owed or to extend, delete, or modify their locked positions in unintended ways, and any risk of out-of-gas exceptions due to high gas consumption in certain functions. Our investigation revealed that a critical storage variable is incorrectly computed, resulting in the incorrect accounting of expired user positions (TOB-LSK2-9) and that under certain circumstances certain functions could consume a huge amount of gas (TOB-LSK2-12).

    In addition to these findings, the Lisk team identified an issue that would have caused incorrect locked positions to be updated when the `increaseLockingAmount` function is called with multiple positions. This is because the function always uses the first index of the provided array instead of iterating over all of the items in the array. The Lisk team fixed the issue during the engagement (see the Fix Review appendix for more detail).

- **`L2Airdrop` contract**: The `L2Airdrop` contract is an implementation of the Lisk v4 migration airdrop mechanism on L2. It is responsible for calculating and distributing LSK tokens to the accounts of recipients who have migrated to the L2 network.

  We performed a comprehensive analysis of the `L2Airdrop` contract, looking for potential breaches of access controls, issues that could result in the loss of funds, incorrect verification of the Merkle root, and any abnormal ways that users could claim airdrops. However, our examination did not uncover any issues within this contract.

- **`L2VestingWallet` contract**: This contract handles the vesting functionality of the LSK token for the L2 network. For this contract, our analysis centered on identifying any issues related to the improper initialization of the contract, violations of access controls during the transfer of ownership, and unauthorized ways of upgrading the contract. However, our investigation did not reveal any issues within this contract.

- **The off-chain `claim-backend` and `tree-builder` components**: We manually reviewed the source code of these components to identify any security issues, such as risk of injection, risk of denial of service, and configuration issues (TOB-LSK2-3). We also manually interacted with the `/rpc` endpoint using Burp Suite Professional to see whether an attacker could manipulate the back end or whether the back end exposes any sensitive data (TOB-LSK2-7). Finally, we performed static analysis on the components using Semgrep and CodeQL

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- The `L2Claim`, `L1LiskToken`, `L2LiskToken`, and `Ed25519` contracts were not reviewed, as they were considered out of scope.

- The review of the `tree-builder` and `claim-backend` components focused only on the additions to the code related to the airdrop Merkle tree.

- We assumed that external dependencies work as expected, and we reviewed them only to gain context on their use in the in-scope contracts and to confirm their correct usage.

- While the `L2Reward` contract's internal accounting was reviewed for accounting errors, logic errors, and incorrect state updates, we believe this component would benefit from more robust testing to confirm that combinations of user actions cannot put the system into an invalid or unexpected state.

- The deployment scripts were reviewed for general correctness and potential front-running issues; however, they were considered a lower priority and received a less comprehensive review.

Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in house, to perform automated testing of source code and compiled software.

Test Harness Configuration

We used the following tools in the automated testing phase of this project:

Tool	Description	Policy
Slither	A static analysis framework that can statically verify algebraic relationships between Solidity variables	Appendix E
`slither-mutate`	A deterministic mutation generator that detects gaps in test coverage	Appendix D
Echidna	A smart contract fuzzer that can rapidly test security properties via malicious, coverage-guided test case generation	Appendix E
Semgrep	An open-source static analysis tool for finding bugs and enforcing code standards when editing or committing code and during build time	Appendix E
CodeQL	A code analysis engine developed by GitHub to automate security checks	Appendix E

Areas of Focus

Our automated testing and verification work focused on the following system properties:

- The `updateGlobalState` function never reverts, causing the `L2Reward` contract to freeze user assets.
- Users are not able to bypass the maximum lock duration limit to mint more voting power tokens.
- The `totalWeights` accounting value is correct for each day.

Additionally, we used automated tooling to do the following:

- Discover coverage gaps in the testing of smart contracts
- Look for common issues in the smart contracts and the back-end components

Test Results

The results of this focused testing are detailed below.

`L2Reward`: This contract allows users to create and modify locked positions and earn rewards over time.

Property	Tool	Result
The `updateGlobalState` function never reverts.	Echidna	**Passed**
Users cannot create a position whose duration is larger than `MAX_LOCKING_DURATION`.	Echidna	**Passed**
The sum of all position weights during a time period is equal to the sum of all `totalWeights` state variables during the same period.	Echidna	**Further Investigation Required**

`slither-mutate`: The following table displays the portion of each type of mutant for which all unit tests passed. The presence of valid mutants indicates that there are gaps in test coverage because the test suite did not catch the introduced change.

- Uncaught revert mutants replace a given expression with a `revert` statement and indicate that the line is not executed during testing.
- Uncaught comment mutants comment out a given expression and indicate that the effects of this line are not checked by any assertions.
- Uncaught tweak mutants indicate that the expression being executed features edge cases that are not covered by the test suite.

The `src/L2` subdirectory is the root for all target paths listed below. Contracts supporting tests or that produced zero analyzed mutants (e.g., interfaces) were omitted from mutation testing analysis.

Target	Uncaught Reverts	Uncaught Comments	Uncaught Tweaks
`L2VotingPower`	**0%**	**19%**	**19.3%**

	(0/16)	(4/21)	(11/57)
L2Governor	33.3% (7/21)	29.4% (5/17)	50% (6/12)
L2Staking	1.2% (1/79)	17.4% (12/69)	13.4% (34/254)
L2Reward	0% (0/133)	8.4% (10/119)	15.3% (136/891)
L2LockingPosition	0% (0/45)	11.4% (4/35)	17.3% (18/104)
L2VestingWallet	0% (0/7)	40% (4/10)	0% (0/2)

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The codebase uses a modern version of the Solidity compiler that includes overflow protection by default, and the codebase contains no instances of unchecked arithmetic. The arithmetic formulas are documented; however, the `L2Reward` contract's internal accounting would benefit from additional specification and more thorough testing, as indicated by the results of the mutation testing campaign (see appendix D for more detail).	**Moderate**
Auditing	Most state-changing functions emit events; however, we identified several functions in the `L2Reward` and `L2Staking` contracts that do not (TOB-LSK2-10). More consistent event emission would make it easier for the Lisk team and the contract users to monitor the state of the contracts. Also, we are not aware of any monitoring system or incident response plan in place.	**Weak**
Authentication / Access Controls	The system uses a two-step process for crucial state-changing operations such as transferal of contract ownership. The system's access controls are robust, and we identified no access control issues during the review. However, all privileged actions are callable by a single actor, a 3-of-5 multisignature wallet. Creating detailed user documentation on the nature of this privileged actor and when certain privileged actions will be taken would be beneficial.	**Moderate**
Complexity Management	The system includes multiple contracts that are tightly interwoven. While the system architecture is simple and the components are well defined, we identified multiple instances of code duplication, which makes the system	**Moderate**

	more difficult to maintain and makes it easier to introduce bugs during code updates. The system's code duplication could be eliminated through the addition of a helper contract or library that houses reused logic.	
Configuration	Even though the `DB_HOST` environment variable is configurable, the Docker Compose configuration does not pin the PostgreSQL service to a specific host (TOB-LSK2-8). Additionally, the server certificate validation is completely disabled (TOB-LSK2-3).	**Moderate**
Decentralization	The contracts are upgradeable by a single actor (the contract owner), and all privileged actions are callable by this actor. Additionally, users have no way of opting out of contract upgrades. Users could benefit from documentation clearly identifying any system risks, edge cases, and considerations they should keep in mind when choosing to interact with the system.	**Weak**
Documentation	The contracts contain inline and NatSpec documentation that makes it easy to understand the developer's intentions. Inline code comments are descriptive and detailed, providing valuable insights. The majority of the documentation is thorough and effectively clarifies the functionality of the code.	**Satisfactory**
Low-Level Manipulation	The in-scope contracts do not use assembly or low-level calls. While bitwise operations are used in the `L2Airdrop` contract, we identified no issues related to these operations.	**Satisfactory**
Maintenance	While our assessment did not prioritize checking for outdated third-party dependencies, our analysis of certain components revealed some vulnerable libraries (TOB-LSK2-6); however, our review suggests they are not exploitable.	**Satisfactory**
Memory Safety and Error Handling	Exception handling is implemented in the off-chain components, but sometimes it is inconsistent (see the third recommendation provided in appendix C). Introducing TypeScript fuzzing into the CI/CD process would help identify potential uncaught exceptions. Off-chain components are based on TypeScript; this	**Satisfactory**

	reduces their exposure to memory safety issues.	
Testing and Verification	The codebase contains Foundry unit tests with good coverage over the in-scope contracts. However, mutation testing discovered significant coverage gaps in the `L2Reward` contract accounting tests, as well as minor gaps in tests of state updates and edge cases in other system contracts. The system would benefit from an expansion of its unit tests and the addition of fuzzing tests (see appendix D for more detail). Because the TypeScript code relies heavily on parsing data, consider implementing fuzzing of the TypeScript code—for example, using the Jazzer.js fuzzer in Jest tests. Fuzzing will help find unexpected scenarios and uncaught exceptions.	**Moderate**
Transaction Ordering	The codebase is generally not susceptible to timing attacks; however, the OpenZeppelin `Governor` contract could be susceptible to proposal front-running if users omit some optional parameters. Creating user documentation that clearly informs users of the importance of these parameters could help prevent such cases.	**Satisfactory**

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Users can bypass the minimum lock duration	Data Validation	Low
2	Removing L2Reward from allowedCreators will freeze all positions created through the contract	Denial of Service	Low
3	Missing certificate validation	Cryptography	Medium
4	Synchronous function calls inside asynchronous functions	Denial of Service	Informational
5	Hard-coded credentials	Data Exposure	Low
6	Use of outdated libraries	Patching	Informational
7	Stack traces in Express are not disabled	Data Exposure	Informational
8	Docker Compose ports exposed on all interfaces	Configuration	Informational
9	Extending the duration of an expired position can break protocol accounting	Data Validation	Medium
10	Insufficient event generation	Auditing and Logging	Informational
11	Users are charged a larger penalty for fast unlocks than necessary	Data Validation	Informational
12	Potential for huge gas consumption in updateGlobalState and calculateRewards	Data Validation	Informational

Detailed Findings

1. Users can bypass the minimum lock duration

Severity: **Low**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-LSK2-1
Target: `src/L2/L2Staking.sol`	

Description

Users can bypass the 14-day minimum lock duration by depositing a small amount of LSK tokens, initiating a fast unlock of their positions, and then depositing more tokens.

The `L2Staking` contract allows users to initiate a fast unlock of their positions in order to be able to withdraw their locked LSK tokens early (figure 1.1). This action charges a penalty to the user proportional to their locked amount and the remaining lock duration, and sets the position's lock duration to three days (figure 1.2).

```
function initiateFastUnlock(uint256 lockId) public virtual returns (uint256) {
    IL2LockingPosition.LockingPosition memory lock =
        (IL2LockingPosition(lockingPositionContract)).getLockingPosition(lockId);
    require(isLockingPositionNull(lock) == false, "L2Staking: locking position does
not exist");
    require(canLockingPositionBeModified(lockId, lock), "L2Staking: only owner or
creator can call this function");
    require(remainingLockingDuration(lock) > FAST_UNLOCK_DURATION, "L2Staking: less
than 3 days until unlock");

    // calculate penalty
    uint256 penalty = calculatePenalty(lock.amount, remainingLockingDuration(lock));

    uint256 amount = lock.amount - penalty;
    uint256 expDate = todayDay() + FAST_UNLOCK_DURATION;

    // update locking position
    (IL2LockingPosition(lockingPositionContract)).modifyLockingPosition(lockId,
amount, expDate, 0);

    if (lock.creator == address(this)) {
        // send penalty amount to the Lisk DAO Treasury contract
        bool success = IERC20(l2LiskTokenContract).transfer(daoTreasury, penalty);
        require(success, "L2Staking: LSK token transfer from Staking contract to DAO
failed");
    } else {
```

```
        // send penalty amount to the creator
        bool success = IERC20(l2LiskTokenContract).transfer(lock.creator, penalty);
        require(success, "L2Staking: LSK token transfer from Staking contract to
creator failed");
    }

    return penalty;
}
```

Figure 1.1: The `initiateFastUnlock` function in L2Staking.sol#L285–312

```
function calculatePenalty(uint256 amount, uint256 remainingDuration) internal view
virtual returns (uint256) {
    if (emergencyExitEnabled) {
        return 0;
    }

    return (amount * remainingDuration) / (MAX_LOCKING_DURATION *
PENALTY_DENOMINATOR);
}
```

Figure 1.2: The `calculatePenalty` function in L2Staking.sol#L142–148

However, users that have initiated a fast unlock are still able to add to their position's locked amount through the `increaseLockingAmount` function, since this function verifies only that the position is not expired (figure 1.3).

```
function increaseLockingAmount(uint256 lockId, uint256 amountIncrease) public
virtual {
    IL2LockingPosition.LockingPosition memory lock =
        (IL2LockingPosition(lockingPositionContract)).getLockingPosition(lockId);
    require(isLockingPositionNull(lock) == false, "L2Staking: locking position does
not exist");
    require(canLockingPositionBeModified(lockId, lock), "L2Staking: only owner or
creator can call this function");
    require(amountIncrease > 0, "L2Staking: increased amount should be greater than
zero");
    require(
        lock.pausedLockingDuration > 0 || lock.expDate > todayDay(),
        "L2Staking: can not increase amount for expired locking position"
    );
    [...]
}
```

Figure 1.3: The `increaseLockingAmount` function in L2Staking.sol#L317–338

Users could exploit this validation issue to bypass the minimum 14-day lock duration. Specifically, a user could deposit a very small amount and then initiate a fast unlock of the position, reducing the position's lock duration to three days for a negligible penalty. The user could then call `increaseLockingAmount` to deposit the larger amount of funds they intended to deposit, leaving with a lock duration of only three days for very little cost.

Exploit Scenario

Alice wants to deposit 100,000 LSK tokens in order to gain a large amount of voting power; however, she does not want to have to lock her position for 14 days. She takes the following actions to reduce her lock duration:

1. Deposits the minimum amount of LSK tokens (0.01 LSK) for the minimum duration (14 days)
2. Initiates a fast unlock of her position and pays a small penalty (0.0028 LSK) to reduce the position's lock duration to three days
3. Deposits 100,000 LSK to her position

Her position is now locked for three days instead of the minimum 14 days, and she has paid a negligible penalty for this benefit.

Recommendations

Short term, add a check to the `increaseLockingAmount` function that ensures positions that have a lock duration of less than the minimum cannot be increased.

Long term, define a list of function- and system-level invariants and use advanced testing techniques such as smart contract fuzzing with Echidna, Medusa, or Foundry invariant testing to verify that the invariants hold.

2. Removing L2Reward from allowedCreators will freeze all positions created through the contract

Severity: **Low**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-LSK2-2
Target: `src/L2/L2Staking.sol`	

Description

If the `L2Reward` contract is removed from the `L2Staking` contract's `allowedCreators` mapping, all positions that were created through `L2Reward` will be frozen until it is re-added.

The owner of the `L2Staking` contract is able to add and remove addresses from the `allowedCreators` mapping at any time (figure 2.1). This feature enables users to create their positions through an external contract (i.e., the `L2Reward` contract) and prevents them from modifying their positions by directly calling the `L2Staking` contract. This is done to ensure that the internal accounting of the `L2Reward` contract is always correct and in sync with the `L2Staking` contract's internal state.

```
function addCreator(address newCreator) public virtual onlyOwner {
    require(newCreator != address(0), "L2Staking: creator address can not be zero");
    require(newCreator != address(this), "L2Staking: Staking contract can not be
added as a creator");
    allowedCreators[newCreator] = true;
    emit AllowedCreatorAdded(newCreator);
}

function removeCreator(address creator) public virtual onlyOwner {
    require(creator != address(0), "L2Staking: creator address can not be zero");
    delete allowedCreators[creator];
    emit AllowedCreatorRemoved(creator);
}
```

Figure 2.1: The `addCreator` and `removeCreator` functions in `L2Staking.sol#L192–206`

All functions in the `L2Staking` contract use the `canLockingPositionBeModified` internal function as an access control mechanism. If a position was created through an external contract that is in the `allowedCreators` mapping, the position can be modified only by that contract. Further access controls are implemented in the `L2Reward` contract to ensure that only the owner of a position can modify it through the contract.

```
function canLockingPositionBeModified(
    uint256 lockId,
    IL2LockingPosition.LockingPosition memory lock
)
    internal
    view
    virtual
    returns (bool)
{
    address ownerOfLock =
(IL2LockingPosition(lockingPositionContract)).ownerOf(lockId);
    bool condition1 = allowedCreators[msg.sender] && lock.creator == msg.sender;
    bool condition2 = ownerOfLock == msg.sender && lock.creator == address(this);

    if (condition1 || condition2) {
        return true;
    }
    return false;
}
```

Figure 2.2: The `canLockingPositionBeModified` function in `L2Staking.sol#L119–136`

However, this feature also gives the contract owner the power to freeze all positions created through the `L2Reward` contract by simply removing the `L2Reward` contract address from the `allowedCreators` mapping. This issue applies to any address added to the `allowedCreators` mapping in the future.

Exploit Scenario

Alice creates a locked position by depositing 1 million LSK tokens through the `L2Reward` contract. The owner of the `L2Staking` contract removes the `L2Reward` contract from the `allowedCreators` mapping, preventing Alice and all other users who have created positions through `L2Reward` from modifying their positions and accessing their tokens.

Recommendations

Short term, implement an emergency withdrawal function that can be used only in the event that the `L2Reward` contract is removed from the `allowedCreators` mapping.

Long term, create user-facing documentation outlining the powers of privileges of actors and all risks associated with interacting with the contracts.

3. Missing certificate validation

Severity: **Medium**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-LSK2-3
Target: `lisk-token-claim-private/packages/claim-backend/src/db.ts`	

Description

The client-side verification of the server certificate is disabled in the DB class, allowing for server impersonation and person-in-the-middle attacks when the `DB_SSLMODE` environment variable is set to `true` (figure 3.1).

```
20    process.env.DB_SSLMODE === 'true'
21          ? {
22                        ssl: {
23                               require: true,
24                               rejectUnauthorized: false,
25                        },
26                }
27          : {},
```

Figure 3.1: The setup of TLS in the DB class's constructor (lisk-token-claim/packages/claim-backend/src/db.ts#20–27)

Exploit Scenario

An attacker poses as the PostgreSQL server and presents a fake certificate. Because verification of the server certificate is disabled, the attacker's certificate is accepted, allowing him to interfere with communication.

Recommendations

Short term, re-enable TLS certificate verification in the DB class constructor. Review the TLS configuration to ensure it uses modern TLS protocol versions and ciphers.

Long term, incorporate the Semgrep tool with the `bypass-tls-verification` rule in the CI/CD process to catch issues like this early on.

4. Synchronous function calls inside asynchronous functions	
Severity: **Informational**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-LSK2-4
Target: `tree-builder`	

Description

In multiple locations in the `tree-builder` component, synchronous functions are called inside asynchronous functions; for example, the synchronous `fs.writeFileSync` function is called inside the asynchronous `createKeyPairs` function (figure 4.1). Such calls will block the event loop and essentially defeat the purpose of asynchronicity. Since the event loop is single-threaded, blocking it will prevent all other asynchronous tasks from running concurrently. If an attacker can cause large files to be written, other users could be prevented from taking actions in the system, causing a denial of service.

```
  7    export async function createKeyPairs(amount = 100) {
// (...)
 21          fs.writeFileSync('../../data/example/key-pairs.json',
JSON.stringify(keys), 'utf-8');
 22    }
```

Figure 4.1: A call to the `fs.writeFileSync` function inside of the async `createKeyPairs` function (lisk-token-claim/packages/tree-builder/src/applications/example/create_key_pairs.ts#7–22)

Synchronous functions are called inside asynchronous functions in these other locations as well:

- tree-builder/src/applications/generate-airdrop-merkle-tree/build_airdrop_tree_json.ts#L14-L14
- tree-builder/src/applications/generate-airdrop-merkle-tree/build_airdrop_tree_json.ts#L29-L29
- tree-builder/src/applications/generate-airdrop-merkle-tree/build_airdrop_tree_json.ts#L48-L48
- tree-builder/src/applications/generate-merkle-tree/build_tree_json.ts#L14-L14

- tree-builder/src/applications/generate-merkle-tree/build_tree_json.ts#L25-L25
- tree-builder/src/applications/generate-merkle-tree/build_tree_json.ts#L43-L43
- tree-builder/src/commands/example/index.ts#L42-L42
- tree-builder/src/commands/generate-airdrop-merkle-tree/index.ts#L103-L103
- tree-builder/src/commands/generate-airdrop-merkle-tree/index.ts#L70-L70
- tree-builder/src/commands/generate-merkle-tree/index.ts#L52-L52
- tree-builder/src/commands/generate-merkle-tree/index.ts#L82-L82
- tree-builder/src/commands/generate-merkle-tree/index.ts#L85-L85

Recommendations

Short term, replace all calls to synchronous functions inside asynchronous functions with their asynchronous equivalents.

Long term, use the Semgrep tool with the custom Semgrep rule provided in appendix F in the CI/CD process to find any other instances of this issue.

5. Hard-coded credentials

Severity: **Low**	Difficulty: **Low**
Type: Data Exposure	Finding ID: TOB-LSK2-5
Target: `lisk-token-claim/packages/claim-backend/docker-compose.yml`, `lisk-token-claim/packages/claim-backend/src/db.ts`	

Description

The `claim-backend` component sets the PostgreSQL password as an environment variable (figure 5.1). The password is also checked into the source code (figure 5.2).

Keeping secrets in environment variables is a well-known antipattern. Environment variables are commonly captured by all manner of debugging and logging information, can be accessed from `procfs`, and are passed down to all child processes.

If attackers have access to the application source code, they would have access to the database password. Additionally, because the password is checked into the source code repository, all employees and contractors with access to the repository have access to the password. Credentials should never be stored in plaintext in source code repositories, as they can become valuable tools to attackers if the repositories are compromised.

```
 2    services:
 3      postgres:
 4        image: postgres
// (...)
 9        environment:
10          POSTGRES_USER: claim-backend
11          POSTGRES_PASSWORD: passwd
```

Figure 5.1: The PostgreSQL password set as an environment variable (lisk-token-claim/packages/claim-backend/docker-compose.yml#2–11)

```
 8    constructor() {
 9          this.models = [Signature];
10          this.sequelize = new Sequelize({
11                dialect: 'postgres',
12                host: process.env.DB_HOST || '127.0.0.1',
13                database: process.env.DB_DATABASE || 'claim-backend',
14                username: process.env.DB_USERNAME || 'claim-backend',
15                password: process.env.DB_PASSWORD || 'passwd',
```

Figure 5.2: The database password is taken from the DB_PASSWORD environment variable or set to passwd if there is no environment variable. (`lisk-token-claim/packages/claim-backend/src/db.ts#8–15`)

Exploit Scenario

An attacker obtains a copy of the source code from a former employee. He extracts the PostgreSQL password and uses it to exploit Lisk's products.

Recommendations

Short term, use a Docker secret in place of the environment variable for the PostgreSQL password, and remove the password from the source code; use Docker secrets instead of environment variables to pass any other sensitive information into containers moving forward.

Long term, consider storing credentials in a secret management solution. Also, periodically use the TruffleHog tool to detect secrets checked into the source code.

References

- GitHub Docs: Removing sensitive data from a repository

6. Use of outdated libraries

Severity: **Informational**	Difficulty: **High**
Type: Patching	Finding ID: TOB-LSK2-6
Target: `lisk-token-claim`	

Description

We used `npm audit` to detect the use of outdated dependencies in the `lisk-token-claim` component, which identified the following vulnerable packages.

Dependency	Vulnerability Report	Vulnerability Description	Vulnerable Versions
`ejs`	CVE-2024-33883	`ejs` lacks pollution protection	< 3.1.10
`lodash` (used in `oclif/plugin-warn-if-update-available`)	CVE-2021-23337	`lodash` is vulnerable to command injection via the `template` function	1.7.0 || 2.0.0 || >= 2.1.0

Table 6.1: Vulnerable dependencies used in `lisk-token-claim`

Recommendations

Short term, update `ejs`, `lodash`, and all other build process dependencies to their latest versions wherever possible. Use tools such as `npm audit` and `yarn audit` to confirm that no vulnerable dependencies remain.

Long term, implement these checks as part of the project's CI/CD pipeline.

7. Stack traces in Express are not disabled

Severity: **Informational**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-LSK2-7
Target: `claim-backend`	

Description

The `NODE_ENV` environment variable is not set to `production` in the `claim-backend` component. As a result, the underlying Express web framework returns errors to the client along with a stack trace (figure 7.1), which exposes internal `claim-backend` paths and functions. (Express does not return stack traces in the production environment.)

```
POST /rpc HTTP/1.1
Host: 127.0.0.1:3000
Content-Type: application/json
Content-Length: 161

{
    "jsonrpc": "2.0",
    "method": "checkEligibility",
    "params": {
        "lskAddress": "lskfcu7z7sch46o67sq24v9h9df2h5o2juvjp3fjj"
    },
    "id": 1
}'

HTTP/1.1 400 Bad Request
// (...)
<pre>SyntaxError: Unexpected token &#39; in JSON at position 160<br>    at
JSON.parse (&lt;anonymous&gt;)<br>    at parse
(/node_modules/body-parser/lib/types/json.js:92:19)<br>    at
/node_modules/body-parser/lib/read.js:128:18<br>    at
AsyncResource.runInAsyncScope (node:async_hooks:203:9)<br>    at
invokeCallback (/node_modules/raw-body/index.js:238:16)<br>    at done
(/node_modules/raw-body/index.js:227:7)<br>    at IncomingMessage.onEnd
(/node_modules/raw-body/index.js:287:7)<br>    at IncomingMessage.emit
(node:events:517:28)<br>    at endReadableNT
(node:internal/streams/readable:1400:12)<br>    at
process.processTicksAndRejections (node:internal/process/task_queues:82:21)</pre>
</body></html>
```

Figure 7.1: An HTTP request that results in the `400 Bad Request` error and a stack trace

Exploit Scenario

An attacker uses the stack traces returned with errors from the Express web framework to identify internal paths and functions in the `claim-backend` component. This information allows him to prepare further exploits that target the `claim-backend` component.

Recommendations

Short term, set the `NODE_ENV` variable to `production` and check that stack traces are not returned (for example, when a request with intentionally malformed JSON in the body is sent to RPC).

References

- Express: Error Handling

8. Docker Compose ports exposed on all interfaces

Severity: **Informational**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-LSK2-8
Target: `lisk-token-claim/packages/claim-backend/docker-compose.yml`	

Description

Docker ports are specified in the `docker-compose.yml` configuration file without a host. For example, `${DB_PORT}:5432` is specified for the PostgreSQL container (figure 8.1). This means that these ports are accessible not just to other processes running on the same computer, but also to other computers on the same network.

```
2    services:
3      postgres:
4        image: postgres
5        restart: always
6        ports:
7          - '${DB_PORT}:5432'
```

Figure 8.1: The PostgreSQL port is exposed on all interfaces.
(lisk-token-claim/packages/claim-backend/docker-compose.yml#2–7)

Recommendations

Short term, set all configuration values in `docker-compose.yml` with both hosts and ports. For example, set the PostgreSQL port to `${DB_HOST}:${DB_PORT}:5432` instead of `${DB_PORT}:5432`.

Long term, use the `port-all-interfaces` Semgrep rule to detect and flag instances of this configuration pattern.

References

- Docker Docs: Networking in Compose

9. Extending the duration of an expired position can break protocol accounting

Severity: **Medium**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-LSK2-9
Target: `src/L2/L2Reward.sol`	

Description

An incorrect `dailyUnlockedAmounts` mapping key is used in the `_extendDuration` function. As a result, extending an expired position can break protocol accounting.

The `L2Reward` contract allows users to extend the duration of their locked positions by passing an array of lock IDs to the `extendDuration` function, which calls the internal `_extendDuration` function for each lock ID. This function updates different global accounting values depending on whether the given position has expired (figure 9.1).

```
function _extendDuration(uint256 lockID, uint256 durationExtension) internal virtual
{
    IL2LockingPosition.LockingPosition memory lockingPosition =
        IL2LockingPosition(lockingPositionContract).getLockingPosition(lockID);

    // claim rewards and update staking contract
    _claimReward(lockID);
    IL2Staking(stakingContract).extendLockingDuration(lockID, durationExtension);

    // update globals
    totalWeight += lockingPosition.amount * durationExtension;

    if (lockingPosition.pausedLockingDuration == 0) {
        // locking period has not finished
        if (lockingPosition.expDate > todayDay()) {
            dailyUnlockedAmounts[lockingPosition.expDate] -= lockingPosition.amount;
        }
        // locking period has expired, re-lock amount
        else {
            totalAmountLocked += lockingPosition.amount;
            pendingUnlockAmount += lockingPosition.amount;
            totalWeight += lockingPosition.amount * OFFSET;
        }

        dailyUnlockedAmounts[lockingPosition.expDate + durationExtension] +=
lockingPosition.amount;
    }
}
```

Figure 9.1: The `_extendDuration` function in L2Reward.sol#L394–L419

The highlighted line in figure 9.1 shows that the `_extendDuration` function uses the sum of the previous expiration date of the given position and the value of the `durationExtension` input variable as the `dailyUnlockedAmounts` mapping key. However, based on the `extendLockingDuration` function of the `L2Staking` contract, it is clear that the key used in `_extendDuration` is incorrect (figure 9.2).

```
function extendLockingDuration(uint256 lockId, uint256 extendDays) public virtual {
    [...]

    if (lock.pausedLockingDuration > 0) {
        // remaining duration is paused
        lock.pausedLockingDuration += extendDays;
    } else {
        // remaining duration not paused, if expired, assume expDate is today
        lock.expDate = Math.max(lock.expDate, todayDay()) + extendDays;
    }

    [...]
}
```

Figure 9.2: The `extendLockingDuration` function in L2Staking.sol#L354–360

If the position has expired and the sum of `lockingPosition.expDate` and `durationExtension` is less than or equal to the value of `todayDay`, the `dailyUnlockedAmounts` mapping will use a key that was already used to update the global state in the past. As a result, the values of the global state variables `pendingUnlockAmount`, `totalAmountLocked`, `totalWeight`, and `cappedRewards` will be incorrect.

This can have the following consequences:

- The `dailyRewards` limit could be incorrectly applied.
- User rewards could be reduced if the `totalWeight` value is larger than it should be.
- The `updateGlobalState` function could revert, freezing all user positions and assets.

Exploit Scenario

Alice deposits 10e18 LSK tokens, creating a position with a lock duration of 14 days. On day 16, she decides to extend her position by one day. Although her position is extended until day 17, the `dailyUnlockedAmounts` value is updated for day 15, causing the `pendingUnlockAmount`, `totalAmountLocked`, and `totalWeight` accounting values to be incorrect.

Recommendations

Short term, fix the highlighted line of figure 9.1 by adding the line `lockingPosition.expDate = Math.max(lockingPosition.expDate, todayDay())` to the `else` branch, above the highlighted line.

Long term, improve the testing suite by adding unit and fuzz tests that verify that the values of the global state variables are correct.

10. Insufficient event generation

Severity: **Informational**	Difficulty: **Low**
Type: Auditing and Logging	Finding ID: TOB-LSK2-10
Target: `src/L2/L2Reward.sol`, `src/L2/L2Staking.sol`	

Description

Multiple user operations do not emit events. As a result, it will be difficult to review the contracts' behavior for correctness once they have been deployed.

Events generated during contract execution aid in monitoring, baselining of behavior, and detection of suspicious activity. Without events, users and blockchain-monitoring systems cannot easily detect behavior that falls outside the baseline conditions; malfunctioning contracts and attacks could go undetected.

The following operations should trigger events:

- `L2Reward`:
  - `createPosition`
  - `deletePosition`
  - `initiateFastUnlock`
  - `increaseLockingAmount`
  - `extendDuration`
  - `pauseLocking`
  - `resumeLockingCountdown`
- `L2Staking`:
  - `lockAmount`
  - `unlock`
  - `initiateFastUnlock`
  - `increaseLockingAmount`
  - `extendLockingDuration`

- `pauseRemainingLockingDuration`
- `resumeCountdown`

Exploit Scenario
An attacker discovers a vulnerability in the `L2Reward` contract and modifies its execution. Because these actions generate no events, the behavior goes unnoticed until there is follow-on damage, such as financial loss.

Recommendations
Short term, add events for all operations that could contribute to a higher level of monitoring and alerting.

Long term, consider using a blockchain-monitoring system to track any suspicious behavior in the contracts. The system relies on several contracts to behave as expected. A monitoring mechanism for critical events would quickly detect any compromised system components.

11. Users are charged a larger penalty for fast unlocks than necessary

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-LSK2-11
Target: `src/L2/L2Staking.sol`	

Description

The penalty calculation in the fast unlocking mechanism is incorrect and will charge users a higher penalty than what they truly owe.

As shown in figure 11.1, the `initiateFastUnlock` function in the `L2Staking` contract allows users to reduce the lock duration of their positions to three days. Users incur a penalty for this action proportional to their locked amount and the remaining lock duration.

```
function initiateFastUnlock(uint256 lockId) public virtual returns (uint256) {
    ...
    // calculate penalty
    uint256 penalty = calculatePenalty(lock.amount, remainingLockingDuration(lock));

    uint256 amount = lock.amount - penalty;
    uint256 expDate = todayDay() + FAST_UNLOCK_DURATION;
    ...
}
```

Figure 11.1: The `initiateFastUnlock` function in L2Staking.sol#L293–296

However, the associated penalty calculation computes the penalty based on the remaining days of the lock duration from when the user initiates the fast unlock request, including the three-day period during which the position will still be locked.

Recommendations

Short term, replace the penalty calculation logic with the following:

```
uint256 penalty = calculatePenalty(lock.amount,
remainingLockingDuration(lock) - FAST_UNLOCK_DURATION);
```

Long term, perform a thorough review of the code to identify any potential logical calculation errors in the future.

12. Potential for huge gas consumption in updateGlobalState and calculateRewards

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-LSK2-12
Target: `src/L2/L2Reward.sol`	

Description

Under certain conditions, the `updateGlobalState` and `calculateRewards` functions in the `L2Reward` contract can consume a substantial amount of gas.

First, the `calculateRewards` function calculates a user's rewards by looping over the period between the day the user created the position and the day the user claims their rewards (figure 12.1). The larger this period is, the more gas is consumed. If this period is sufficiently large, the loop could cause an out-of-gas exception.

```
function calculateRewards(uint256 lockID) public view virtual returns (uint256) {
    ...
    for (uint256 d = lastClaimDate[lockID]; d < lastRewardDay; d++) {
        reward += (weight * dailyRewards[d]) / totalWeights[d];

        if (lockingPosition.pausedLockingDuration == 0) {
            // unlocking period is active, weight is decreasing
            weight -= lockingPosition.amount;
        }
    }
    return reward;
}
```

Figure 12.1: The `calculateRewards` function in `L2Reward.sol#L249–287`

The `updateGlobalState` function could consume a significant amount of gas for a similar reason. The loop iteration in this function depends on the period between the date of the last transaction and the current day (figure 12.2). As a result, the greater the duration, the more expensive the loop becomes and the more gas is consumed. As with the `calculateRewards` function, a sufficiently long duration could cause an out-of-gas exception.

```
function updateGlobalState() public virtual {
    uint256 today = todayDay();

    uint256 d = lastTrsDate;
```

```
    if (today <= d) return;

    uint256 cappedRewards;

    for (; d < today; d++) {
        totalWeights[d] = totalWeight;

        cappedRewards = totalAmountLocked / 365;
        ...
    }
    lastTrsDate = today;
}
```

Figure 12.2: The `updateGlobalState` function in `L2Reward.sol#L116–145`

Recommendations

Short term, periodically invoke the `updateGlobalState` function to keep the date of the last transaction updated. This will ensure the period between the last transaction date and the current date never becomes excessively long. Add the gas consumption concerns for the `calculateRewards` function to the user-facing documentation so that users are aware of the risks.

Long term, maintain regular monitoring of the gas usage of these functions to detect instances in which they consume a significant amount of gas.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Configuration	The configuration of system components in accordance with best practices
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Maintenance	The timely maintenance of system components to mitigate risk
Memory Safety and Error Handling	The presence of memory safety and robust error-handling mechanisms
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage
Transaction Ordering	The system's resistance to transaction-ordering attacks

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Code Quality Recommendations

This appendix contains recommendations for findings that do not have immediate or obvious security implications. However, these issues may facilitate exploit chains targeting other vulnerabilities, become easily exploitable in future releases, or decrease code readability.

- **In the `loadAirdropMerkleTree` and `loadMerkleTree` functions, add a check for whether the `leaves` constant is iterable.** If it is not, the functions will crash and return `TypeError: leaves is not iterable`.
- **Add a check for whether there are more than zero accounts to generate in the `buildTree` and `buildAirdropTree` functions.** If not, the functions should return an error early on. Currently, the underlying package throws an `Expected non-zero number of leaves` error.
- **Wrap the parsing functions (such as `readJson`) in a try-catch block to prevent unhandled exceptions.**
- **Replace the line `this.lastClaimDate(lockID)` in the `_claimReward` function of the `L2Reward` contract with the `lastClaimDate` state variable. The current `staticcall` is unnecessary.**
- **Replace instances of `isLockingPositionNull(lock) == false` with `!isLockingPositionNull(lock)`.** This version is more straightforward.
- **Make expiry conditions in the `L2LockingPosition` and `L2Staking`/`L2Reward` contracts consistent.** The `modifyLockingPosition` function of the `L2LockingPosition` contract allows positions with an `expDate` equal to `todayDay()` to be modified; however, the `L2Staking` and `L2Reward` contracts consider such positions to be expired. Expiry conditions should be made consistent between all contracts.
- **Remove extra parentheses surrounding operations casting addresses to interfaces.** Enclosing these operations in parentheses has no effect; they can be removed to reduce visual clutter. For example, both of these versions will behave the same:
  - `(IL2LockingPosition(lockingPositionContract)).createLockingPosition(...)`
  - `IL2LockingPosition(lockingPositionContract).createLockingPosition(...)`

D. Mutation Testing

This appendix outlines how we conducted mutation testing and highlights some of the most actionable results.

At a high level, mutation tests make several changes to each line of a target file and rerun the test suite for each change. Changes that result in test failures indicate adequate test coverage, while changes that do not result in test failures indicate gaps in the test coverage. Although mutation testing is a slow process, it allows auditors to focus their review on areas of the codebase that are most likely to contain latent bugs, and it allows developers to identify and add missing tests.

We used an experimental new mutation tool, `slither-mutate`, to conduct our mutation testing campaign. This tool is custom-made for Solidity and features higher performance and fewer false positives than existing tools such as `universalmutator`.

```
python3 -m pip install slither-analyzer
```

Figure D.1: The command that installs `slither` using `pip`

The mutation campaign was run against the in-scope smart contracts using the following commands:

```
slither-mutate . --test-cmd='forge test' --test-dir='test'
--ignore-dirs='script,lib,test,utils,L1'
```

Figure D.2: A bash script that runs a mutation testing campaign against each Solidity file in the `src` directory

Consider the following notes about the above commands:

- On a consumer-grade laptop, the overall runtime of the mutation testing campaign is approximately four hours.
- The `--test-cmd` flags specify the command to run to assess mutant validity. A `--fail-fast` or `--bail` flag will automatically be added to test commands to improve runtime.

An abbreviated, illustrative example of a mutation test output file is shown in figure D.3.

```
Mutate:Mutating contract L2Reward
INFO:Slither-Mutate:[RR] Line 104: 'require(_l2LiskTokenContract != address(0), "L2Reward: LSK token
contract address can not be zero")' ==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 105: '__Ownable2Step_init()' ==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 106: '__Ownable_init(msg.sender)' ==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 107: '__UUPSUpgradeable_init()' ==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 108: 'l2TokenContract = _l2LiskTokenContract' ==> 'revert()' --> CAUGHT
```

```
INFO:Slither-Mutate:[RR] Line 109: 'lastTrsDate = todayDay()' ==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 110: 'version = "1.0.0"' ==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 111: 'emit LiskTokenContractAddressChanged(address(0x0),
_l2LiskTokenContract)' ==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 124: 'd++' ==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 125: 'totalWeights[d] = totalWeight' ==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 127: 'cappedRewards = totalAmountLocked / 365' ==> 'revert()' -->
CAUGHT
INFO:Slither-Mutate:[RR] Line 131: 'rewardsSurplus += dailyRewards[d] - cappedRewards' ==> 'revert()'
--> CAUGHT
INFO:Slither-Mutate:[RR] Line 132: 'dailyRewards[d] = cappedRewards' ==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 136: 'totalWeight -= pendingUnlockAmount' ==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 137: 'totalWeight -= OFFSET * dailyUnlockedAmounts[d + 1]' ==>
'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 140: 'totalAmountLocked -= dailyUnlockedAmounts[d + 1]' ==> 'revert()'
--> CAUGHT
INFO:Slither-Mutate:[RR] Line 141: 'pendingUnlockAmount -= dailyUnlockedAmounts[d + 1]' ==>
'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 144: 'lastTrsDate = today' ==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 152: 'updateGlobalState()' ==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 155: 'IL2LiskToken(l2TokenContract).transferFrom(msg.sender,
address(this), amount)' ==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 156: 'IL2LiskToken(l2TokenContract).approve(stakingContract, amount)'
==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 159: 'lastClaimDate[id] = today' ==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 162: 'totalWeight += amount * (duration + OFFSET)' ==> 'revert()' -->
CAUGHT
INFO:Slither-Mutate:[RR] Line 163: 'totalAmountLocked += amount' ==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 164: 'dailyUnlockedAmounts[today + duration] += amount' ==> 'revert()'
--> CAUGHT
INFO:Slither-Mutate:[RR] Line 165: 'pendingUnlockAmount += amount' ==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 167: 'return id' ==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 173: 'updateGlobalState()' ==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 175: 'i++' ==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 176: 'require(
                IL2LockingPosition(lockingPositionContract).ownerOf(lockIDs[i]) == msg.sender,
                "L2Reward: msg.sender does not own the locking position"
            )' ==> 'revert()' --> CAUGHT
INFO:Slither-Mutate:[RR] Line 181: '_deletePosition(lockIDs[i])' ==> 'revert()' --> CAUGHT
...
```

Figure D.3: Abbreviated output from the mutation testing campaign assessing test coverage of the in-scope contracts

In summary, the mutation testing campaign caught the following:

- 15 valid mutants in the `L2VotingPower` contract
- 18 valid mutants in the `L2Governor` contract
- 146 valid mutants in the `L2Reward` contract
- 47 valid mutants in the `L2Staking` contract
- 22 valid mutants in the `L2LockingPosition` contract
- 4 valid mutants in the `L2VestingWallet` contract

The following is a summary of the functions and features that lack sufficient tests:

- `L2VotingPower`
  - Changes to line 63 of the `votingPower` function (`position.pausedLockingDuration > 0`) were uncaught, indicating a lack of testing of the voting power calculations for paused and unpaused positions.
  - Changes to lines 92 and 94 of the `adjustVotingPower` function were uncaught, indicating a lack of testing of the code for minting and burning voting power.
- `L2Governor`
  - The governor proposal, queuing, execution, and cancellation operations are untested.
- `L2Staking`
  - Scenarios in which the `canLockingPositionBeModified` function returns `false` are not tested.
  - The `remainingLockingDuration` function is never called on an expired position in tests.
  - The state update for the `lock.pausedLockingDuration` variable in the `extendLockingDuration` function is not tested.
  - A locked position is never extended to `MAX_LOCKING_DURATION` in tests.
- `L2LockingPosition`
  - Scenarios in which the `lockingPositions[positionId]` mapping is deleted in the `removeLockingPosition` function are not tested.
- `L2VestingWallet`
  - The access control of the `transferOwnership` function is not tested.
- `L2Reward`:
  - The `initialize` and `_addRewards` functions are missing tests validating their zero-input checks.
  - The return value of `calculateRewards` is not tested.
  - The early return on line 315 of the `_claimRewards` function is not tested.

- The `_extendDuration` and `_increaseLockingAmount` functions are not tested to ensure that their calls to the `L2Staking` contract have the expected effects.
- The `totalAmountLocked`, `totalWeight`, `dailyUnlockedAmounts`, and `pendingUnlockAmount` state variables are not tested to ensure that they are correctly updated or used in most functions.
- The `addUnusedRewards` function's update of the `rewardsSurplus` state variable is not tested.

Additionally, all implementation contracts are missing tests verifying that they cannot be initialized.

We recommend that the Lisk team review the existing tests and add additional verification that would catch the aforementioned types of mutations. Then, use a script similar to the one provided in figure D.2 to rerun a mutation testing campaign to ensure that the added tests provide adequate coverage.

E. Automated Analysis Tool Configuration

This appendix describes the setup of the automated analysis tools used during this audit.

Though static analysis tools frequently report false positives, they detect certain categories of issues, such as memory leaks, misspecified format strings, and the use of unsafe APIs, with essentially perfect precision. We recommend periodically running these static analysis tools and reviewing their findings.

Slither

We used Slither to detect common issues and antipatterns in the codebase. Although Slither did not discover any severe issues during this review, integrating Slither into the project's testing environment can help find other issues that may be introduced during further development and will help improve the overall quality of the smart contracts' code.

```
slither . --filter-paths '(lib/)'
```

Figure E.1: An example Slither configuration

Integrating `slither-action` into the project's CI pipeline can automate this process.

Echidna

We used Echidna, a smart contract fuzzer that can rapidly test security properties via malicious, coverage-guided test case generation, to check various system states.

```
function durationIsNeverLargerThanMax(uint256 lockID) public {
    lockID = clampBetween(lockID, 1, lastId);
    (,, uint256 expDate, uint256 pausedDuration) =
l2lockingposition.lockingPositions(lockID);
    uint256 maxDuration = l2staking.MAX_LOCKING_DURATION();
    uint256 today = block.timestamp / 1 days;

    assertWithMsg(pausedDuration <= maxDuration, "The position paused duration is
larger than max!");
    if (expDate > today) {
        assertWithMsg(expDate - today <= maxDuration, "The position unpaused
duration is larger than max!");
    }
}
```

Figure E.2: An invariant implementation in the Echidna test harness

```
corpusDir: corpus_E2E
coverage: true
testMode: assertion
codeSize: 0x9000
```

```
testLimit: 5000000
```

Figure E.3: A sample configuration setup in `config.yaml`

```
echidna . --contract EchidnaE2E --config e2e.yaml
```

Figure E.4: The command used to run the Echidna fuzzing campaign

For further guidance on using Echidna, including in-depth documentation and practical examples, visit our Building Secure Contracts website and watch our live streams on Echidna on YouTube.

Semgrep

To install Semgrep, we used `pip` by running `python3 -m pip install semgrep`. Then we installed the Semgrep Pro Engine, which includes cross-file (interfile) and cross-function (interprocedural) analysis.

We ran the following command in the root directory of the project to run our private Trail of Bits rules:

```
semgrep -f /private-semgrep-tob-rules/ --metrics=off
```

Figure E.5: The command used to run our private rules

To run Semgrep with the Pro Engine, we used the following commands:

```
semgrep login
semgrep install-semgrep-pro
```

Figure E.6: The commands used to run the Pro Engine

To run Semgrep on the codebase, we ran the following command in the root directory of all provided source code directories:

```
semgrep --pro -c auto . --sarif -o result.sarif
```

Figure E.7: The command used to run Semgrep on the codebase

CodeQL

We installed CodeQL by following CodeQL's installation guide.

After installing CodeQL, we ran the following command to create the project database for off-chain components:

```
codeql database create codeql.db --language=javascript
```

Figure E.8: The command used to create a project database

We then ran the following command to query the database:

```
codeql database analyze codeql.db -j 10 --format=csv --output=codeql_ts.csv
```

Figure E.9: The command used to query the database

For more information about Semgrep and CodeQL, refer to the Trail of Bits Testing Handbook.

F. Semgrep Rule for TOB-LSK2-4

Figure F.1 contains a Semgrep rule to detect TOB-LSK2-4.

```
rules:
  - id: synchronous-call-in-asynchronous-function
    message: |
      Found synchronous function call inside asynchronous function. This can
      block the NodeJS event loop and lead to denial-of-service under certain
      circumstances. Prefer an equivalent asynchronous function call.
    languages: [javascript, typescript]
    severity: WARNING
    metadata:
      category: best-practice
      cwe: "CWE-400: Uncontrolled Resource Consumption"
      subcategory: [audit]
      confidence: HIGH
      likelihood: HIGH
      impact: LOW
      references:
        - https://nodejs.org/en/guides/dont-block-the-event-loop#blocking-the-event-loop-nodejs-core-modules
        - https://nodejs.org/api/crypto.html
        - https://nodejs.org/api/zlib.html
        - https://nodejs.org/api/child_process.html
        - https://nodejs.org/api/fs.html
    patterns:
      - pattern-inside: async function $FUNC(...) { ... }
      - pattern-either:
          - pattern: crypto.randomBytes($SIZE) # One argument form is sync, two argument form is async
          - pattern: crypto.randomFillSync(...)
          - pattern: crypto.scryptSync(...)
          - pattern: crypto.checkPrimeSync(...)
          - pattern: crypto.generateKeyPairSync(...)
          - pattern: crypto.generateKeySync(...)
          - pattern: crypto.generatePrimeSync(...)
          - pattern: crypto.hkdfSync(...)
          - pattern: crypto.pbkdf2Sync(...)
          - pattern: zlib.brotliCompressSync(...)
          - pattern: zlib.brotliDecompressSync(...)
          - pattern: zlib.deflateSync(...)
          - pattern: zlib.deflateRawSync(...)
          - pattern: zlib.gunzipSync(...)
          - pattern: zlib.gzipSync(...)
          - pattern: zlib.inflateSync(...)
          - pattern: zlib.inflateRawSync(...)
          - pattern: zlib.unzipSync(...)
          - pattern: child_process.execFileSync(...)
          - pattern: child_process.execSync(...)
          - pattern: child_process.spawnSync(...)
          - pattern: fs.accessSync(...)
          - pattern: fs.appendFileSync(...)
          - pattern: fs.chmodSync(...)
          - pattern: fs.chownSync(...)
          - pattern: fs.closeSync(...)
          - pattern: fs.copyFileSync(...)
          - pattern: fs.cpSync(...)
          - pattern: fs.existsSync(...)
          - pattern: fs.fchmodSync(...)
          - pattern: fs.fchownSync(...)
          - pattern: fs.fdatasyncSync(...)
          - pattern: fs.fstatSync(...)
          - pattern: fs.fsyncSync(...)
          - pattern: fs.ftruncateSync(...)
          - pattern: fs.futimesSync(...)
          - pattern: fs.lchmodSync(...)
```

```
        - pattern: fs.lchownSync(...)
        - pattern: fs.lutimesSync(...)
        - pattern: fs.linkSync(...)
        - pattern: fs.lstatSync(...)
        - pattern: fs.mkdirSync(...)
        - pattern: fs.mkdtempSync(...)
        - pattern: fs.opendirSync(...)
        - pattern: fs.openSync(...)
        - pattern: fs.readdirSync(...)
        - pattern: fs.readFileSync(...)
        - pattern: fs.readlinkSync(...)
        - pattern: fs.readSync(...)
        - pattern: fs.readSync(...)
        - pattern: fs.readvSync(...)
        - pattern: fs.realpathSync(...)
        - pattern: fs.realpathSync.native(...)
        - pattern: fs.renameSync(...)
        - pattern: fs.rmdirSync(...)
        - pattern: fs.rmSync(...)
        - pattern: fs.statSync(...)
        - pattern: fs.statfsSync(...)
        - pattern: fs.symlinkSync(...)
        - pattern: fs.truncateSync(...)
        - pattern: fs.unlinkSync(...)
        - pattern: fs.utimesSync(...)
        - pattern: fs.writeFileSync(...)
        - pattern: fs.writeSync(...)
        - pattern: fs.writeSync(...)
        - pattern: fs.writeSync(...)
        - pattern: fs.writevSync(...)
```

Figure F.1: A Semgrep rule to detect TOB-LSK2-4 (`synchronous-call-in-asynchronous-function.yaml`)

G. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

On May 15, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the Lisk team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In addition to reviewing fixes for the issues identified in this report, we also reviewed the fix for an issue identified by the Lisk team, in which the `increaseLockingAmount` function of the `L2Reward` contract incorrectly iterated over the provided positions. The Lisk team addressed this issue by updating the `increaseLockingAmount` function to use the correct index when looping over the input array (PR #116).

In summary, of the 12 issues described in this report, Lisk has resolved 10 issues, has partially resolved one issue, and has not resolved the remaining issue. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Users can bypass the minimum lock duration	Resolved
2	Removing L2Reward from allowedCreators will freeze all positions created through the contract	Resolved
3	Missing certificate validation	Resolved
4	Synchronous function calls inside asynchronous functions	Resolved
5	Hard-coded credentials	Resolved
6	Use of outdated libraries	Unresolved
7	Stack traces in Express are not disabled	Resolved
8	Docker Compose ports exposed on all interfaces	Resolved

9	Extending the duration of an expired position can break protocol accounting	Resolved
10	Insufficient event generation	Resolved
11	Users are charged a larger penalty for fast unlocks than necessary	Resolved
12	Potential for huge gas consumption in updateGlobalState and calculateRewards	Partially Resolved

Detailed Fix Review Results

TOB-LSK2-1: Users can bypass the minimum lock duration
Resolved in PR #97. A check was added to the `increaseLockingAmount` function to ensure that only positions with a remaining lock duration greater than or equal to `MIN_LOCKING_DURATION` can be increased.

TOB-LSK2-2: Removing L2Reward from allowedCreators will freeze all positions created through the contract
Resolved in PR #104. The Lisk team created user-facing documentation outlining the powers of privileged actors and the risks associated with interacting with the contracts.

TOB-LSK2-3: Missing certificate validation
Resolved in PR #36. The `rejectUnauthorized` option is now set to `true` in the DB class, enabling client-side verification of the server certificate.

TOB-LSK2-4: Synchronous function calls inside asynchronous functions
Resolved in PR #39. The synchronous function calls inside asynchronous functions were replaced with their asynchronous equivalents.

TOB-LSK2-5: Hard-coded credentials
Resolved in PR #35. The `claim-backend` component now uses Docker secrets to store passwords rather than environment variables.

TOB-LSK2-6: Use of outdated libraries
Unresolved. The `npm audit` tool still reports vulnerable dependencies. The vulnerable `ejs` package can be fixed by pinning `plugin-help` to the newest version (6.0.22). However, there is still a vulnerable `lodash.template` package in `@oclif/plugin-warn-if-update-available` that is unmaintained and cannot be fixed by the `npm audit fix` tool. Refer to this GitHub issue for more information.

TOB-LSK2-7: Stack traces in Express are not disabled
Resolved. The client provided the following context for this finding's fix status:

Noted and will be using `NODE_ENV=production` in production.

TOB-LSK2-8: Docker Compose ports exposed on all interfaces
Resolved in PR #38. Docker Compose now sets up a port and host based on the `DB_PORT` and `DB_HOST` environment variables, or `127.0.0.1` and `5432` if they are not set.

TOB-LSK2-9: Extending the duration of an expired position can break protocol accounting
Resolved in PR #110. The `_extendDuration` function was updated so that the correct key is used for the `dailyUnlockedAmounts` mapping when updating the accounting of the contract.

TOB-LSK2-10: Insufficient event generation
Resolved in PR #117 and PR #118. Events were added to all of the state-changing functions; each function emits a unique event, making monitoring easier.

TOB-LSK2-11: Users are charged a larger penalty for fast unlocks than necessary
Resolved in PR #107. The `calculatePenalty` function was updated to reduce the penalty by the value of `FAST_UNLOCK_DURATION`, so that users are penalized only for the duration in excess of this value when they initiate a fast unlock.

TOB-LSK2-12: Potential for huge gas consumption in updateGlobalState and calculateRewards
Partially resolved. The client provided the following context for this finding's fix status:

> *Given that we cannot exceed the block gas limit of 30 million, we will maintain the current approach and acknowledge that, in the worst-case scenario, users may incur higher fees. Additionally, we will integrate and use a monitoring tool to observe if significant gaps occur between calls to gas-intensive functions and respond accordingly.*

H. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.