Ai Layer Labs 6079 Smart Contracts

Security Assessment (Summary Report)

May 14, 2024

Prepared for:
Ivan Ravlich
Ai Layer Labs

Prepared by: **Michael Colburn, Alexander Remie, Priyanka Bose, and Suha Hussain**

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:

Anne Marie Barry, Project Manager
annemarie.barry@trailofbits.com

The following engineering director was associated with this project:

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

Michael Brown, Head of Artificial Intelligence/Machine Learning
michael.brown@trailofbits.com

The following consultants were associated with this project:

Michael Colburn, Consultant
michael.colburn@trailofbits.com

Alexander Remie, Consultant
alexander.remie@trailofbits.com

Priyanka Bose, Consultant
priyanka.bose@trailofbits.com

Suha Hussain, Consultant
suha.hussain@trailofbits.com

Project Timeline

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

Date	Event
May 2, 2024	Pre-project kickoff call
May 14, 2024	Delivery of report draft
May 14, 2024	Report readout meeting

Project Targets

The engagement involved a review and testing of the following target.

6079-contracts

Repository	https://github.com/Ai-Layer-Labs/6079-contracts
Version	`03300ae4d56859522890028e58f6ac1785e8a029`
Type	Solidity
Platform	EVM

Executive Summary

Engagement Overview

Ai Layer Labs engaged Trail of Bits to review the security of its 6079 smart contracts at commit `03300ae`. These contracts facilitate the allocation of a reward token to users who stake Lido's stETH in the system on Ethereum and handle the cross-chain communication necessary to actually mint the accrued rewards on Arbitrum.

A team of four consultants conducted the review from May 6 to May 14, 2024 for a total of three engineer-weeks of effort. With full access to source code and documentation, we performed static and dynamic testing of the codebase, using automated and manual processes.

Observations and Impact

Our review of the `Splitter`, `LinearDistributionIntervalDecrease`, `RewardClaimer`, and `Distribution` contracts focused on identifying any issues that could affect the calculation of token shares over time to the different recipient groups, the accuracy of stETH stake accounting, the distribution of rewards to stakers, or introduce errors in the bookkeeping when updating the pools over time. Our review of the `L1Sender`, `L2MessageReceiver`, `L2TokenReceiver`, and `Token` contracts, as well as the `RewardClaimer` and `Distribution` contracts, focused on issues that could prevent minting or token bridging operations from being transmitted correctly, allow a user to transfer or mint more tokens than expected, or improper integration with the Arbitrum, LayerZero, or Uniswap contracts.

The contracts included in the `@layerzerolabs` directory were considered out of scope for this review. The Ai Layers Labs team also noted that they were already aware of the possibility of a negative stETH rebase so we did not prioritize further exploration of this scenario.

Our review resulted in the identification of three issues. One medium-severity and one informational-severity issue were the result of missing input validation of function parameters that could result in funds accidentally being burnt or a confusing user experience. The remaining low-severity issue identified important state changes that omitted on-chains events which could make tracking the system off-chain more difficult.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that Ai Layer Labs take the following steps:

- **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.

- **Continue to improve the documentation, in particular of lower-level implementation details.** Adding more in-line comments to the code, especially in contracts like Splitter and Distribution, will help make the overall system easier to follow.

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 contracts use a modern compiler version that includes built-in overflow protection. The more complex math has been abstracted out to a stateless library which facilitates easier testing. Consistent use of a scaling factor helps to minimize errors due to loss of precision.	**Satisfactory**
Auditing	Most state-changing functions emit events though we did identify several functions that may benefit from additional event coverage (TOB-6079-2). We were not provided with an incident response plan.	**Moderate**
Authentication / Access Controls	There are several explicit roles within the system: each contract's owner, the fee owner, the token minters. The owner capabilities for the `Splitter` and `Distribution` contracts are documented but the remaining contracts and roles would benefit from explicit documentation as well.	**Satisfactory**
Complexity Management	Overall the contracts are broken up into logical pieces. At times the intended functionality can be difficult to follow due to the minimal in-line comments and much of the contract state being tracked in structs which are only defined in the interface contracts so require frequent cross-referencing. Additionally, the way rates are used in some of the contracts is a bit unintuitive as they seem to be closer to an index than a rate, and both the `Splitter` and `Distribution` contracts have a concept of a pool that behave slightly differently, which contribute to making the overall system more difficult to understand.	**Moderate**
Decentralization	With the exception of the token contract, all of the contracts in this system are intended to be upgradeable	**Moderate**

	(though this can be disabled in the `Splitter` and `Distribution` contracts). The token contract's owner controls the minter and who can make changes to the LayerZero OApp configuration. The `Splitter` and `Distribution` contract owners can unilaterally change the rate of future rewards without notice. Documentation suggests that the ownership/governance structure is still a work in progress and may initially be held by a multisig controlled by the foundation, though the configuration of this multisig is unclear. Under normal operating conditions, the allocation and minting of rewards happens programmatically.	
Documentation	The high-level design and developer documentation adequately describe the intended functionality of the system though there are some gaps around lower level implementation details. The contracts themselves have fairly minimal in-line comments. NatSpec comments are limited to externally exposed functions and are in the contract interfaces. Increasing coverage especially of internal functions will help make the codebase's expected functionality easier to understand.	**Moderate**
Low-Level Manipulation	The contracts do not make use of any low-level calls directly. The only in-line assembly use is to unpack the sender address in the `L2MessageReceiver` contract.	**Satisfactory**
Testing and Verification	The contracts have adequate test coverage of positive and negative cases, though they generally rely on hard-coded values. Extending the tests to support a wider range of values and leverage techniques such as fuzzing where possible would be beneficial. The test suite also includes some fork tests and the system documentation includes instructions for manually testing the system on testnets.	**Satisfactory**
Transaction Ordering	We did not identify any transaction ordering vectors that could maliciously affect user funds or the system as a whole.	**Satisfactory**

Summary of Findings

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

ID	Title	Type	Severity
1	Lack of zero-value checks in setter functions	Data Validation	**Informational**
2	Lack of event generation	Auditing and Logging	**Low**
3	Loss of tokens due to lack of zero check in destination address	Data Validation	**Medium**

Detailed Findings

1. Lack of zero-value checks in setter functions	
Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-6079-1
Target: Various	

Description

Certain functions fail to validate incoming arguments, so callers of these functions could mistakenly set important state variables to a zero value, misconfiguring the system.

For example, the `Splitter__init` function in the `Splitter` contract sets `pool` address to the argument `pool_` without checking whether the `pool_` is zero. This may result in undefined behavior in the system.

```
function Splitter__init(Pool memory pool_) external initializer {
    __Ownable_init();
    __UUPSUpgradeable_init();

    _validatePool(pool_);
    pool = pool_;
}
```

Figure 1.1: `Splitter__init` function in `Splitter` contract#L25-31

In addition to the above, the following setter functions also lack zero-value checks:

- `Splitter`
  - `updatePool`
  - `updateGroupShares`
- `Distribution`
  - `Distribution_init`
  - `editPool`
- `L1Sender`
  - `L1Sender__init`
  - `updateAllowedAddresses`
  - `setDepositTokenConfig`
- `L2MessageReceiver`
  - `setParams`

- `L2TokenReceiver`
    - `L2TokenReceiver__init`
- `RewardClaimer`
    - `RewardClaimer__init`
    - `setSplitter`
    - `setL1Sender`

Exploit Scenario

Alice deploys a new version of the Splitter contract. When she invokes to set the `pool` address, she accidentally enters a zero value, thereby misconfiguring the system.

Recommendations

Short term, add zero-value checks to all function arguments to ensure that callers cannot set incorrect values and misconfigure the system.

Long term, use the Slither static analyzer to catch common issues such as this one. Consider integrating a Slither scan into the project's CI pipeline, pre-commit hooks, or build scripts.

2. Lack of event generation	
Severity: **Low**	Difficulty: **Low**
Type: Auditing and Logging	Finding ID: TOB-6079-2
Target: `Various`	

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:

- `Splitter`
    - `Splitter__init`
    - `updatePool`
    - `updateGroupShares`
    - `removeUpgradeability`
- `L1Sender`
    - `L1Sender__init`
    - `setRewardTokenConfig`
    - `setDepositTokenConfig`
    - `updateAllowedAddresses`
    - `_replaceDepositToken`
    - `_replaceDepositTokenGateway`
    - `sendDepositToken`
    - `sendMintMessage`
- `Distribution`
    - `removeUpgradeability`
- `L2MessageReceiver`
    - `setParams`
- `L2TokenReceiver`
    - `editParams`
    - `withdrawToken`
    - `withdrawTokenId`
- `RewardClaimer`

    - `RewardClaimer__init`
    - `setSplitter`
    - `setL1Sender`
- `Token`
    - `updateMinter`

Exploit Scenario
An attacker discovers a vulnerability in any of these contracts 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

3. Loss of tokens due to lack of zero check in destination address

Severity: **Medium**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-6079-3
Target: `Various`	

Description

Certain functions fail to verify whether the destination address is a zero address before sending tokens. This can lead to unintentional burning of tokens.

For example, as shown in figure 3.1,the `claim` function in the `Distribution` contract takes a `receiver` address, which is user-provided, and subsequently uses it to send native tokens via the `sendMintMessage` function in the `L1Sender` contract. Consequently, if the user mistakenly provides a zero address as the `receiver` argument, the tokens will inadvertently be burnt.

```
function claim(address receiver_) external payable {
    address user_ = _msgSender();
    UserData storage userData = usersData[user_];
    ...
    // Transfer rewards
    IL1Sender(l1Sender).sendMintMessage{value: msg.value}(receiver_,
    pendingRewards_, user_);
    emit UserClaimed(user_, receiver_, pendingRewards_);
}
```

Figure 3.1: claim function in Distribution contract#L184-209

In addition to the above, the following function also has similar issues:

- The `claim` function in `RewardClaimer` contract does not verify whether the `receiver_` argument is a zero-address.

Exploit Scenario

Alice invokes the `claim` function in the `Distribution` contract to claim reward tokens. However, by mistake she supplied zero address as the receiver address. Consequently, the claimed rewards are transmitted to the zero address, resulting in an irreversible loss of the tokens.

Recommendations

Short term, add zero-address checks to the function arguments to ensure that callers cannot set incorrect values resulting in loss of tokens.

Long term, use the Slither static analyzer to catch common issues such as this one. Consider integrating a Slither scan into the project's CI pipeline, pre-commit hooks, or build scripts

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
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
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

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

- **Update the `Distribution_init` function to match the other initialization functions.** Most of the upgradeable contracts in the repository follow the pattern `ContractName__init` for naming the initialization function, but the `Distribution` contract's initialization function only uses one underscore instead of two.