× PESSIMISTIC

SECURITY ANALYSIS

by Pessimistic

This report is public

November 11, 2025

ABSTRACT

This report evaluates the security of the smart contracts in the Leea Labs project. The objective was to identify and document security vulnerabilities within the platform's smart contracts.

DISCLAIMER

SUMMARY

This report assesses the security of the Leea Labs smart contracts. The audit process is detailed in the following section.

The audit identified several medium severity issues, including Deposit/withdrawal imbalance at zero TVL and Incorrect price per share calculation at zero TVL. Besides, two low severity issues were found. Also, the developers provided the list of known issues.

After the initial audit, the codebase was updated. The developers resolved two medium-severity issues: Deposit/withdrawal imbalance at zero TVL and Incorrect price per share calculation at zero TVL. They also significantly improved the test suite, although minor issues remain.

The overall quality of the code is mediocre.

NOTE: The **Trader** contracts were excluded from the scope of this audit. These contracts hold deposited assets; therefore, any flaws in their logic could result in a loss of funds for the project.

GENERAL RECOMMENDATIONS

We recommend resolving the remaining issues.

PROJECT OVERVIEW

Project description

For this audit, we reviewed the Leea Labs project from a private GitHub repository, commit 83f4ae937bee300c096761248dae58e486e5ab40.

The scope of the audit included **LeeaVault.sol**.

The provided documentation included the following notion link and README.

The contracts will be deployed to:

- Arbitrum;
- Optimism;
- Gnosis;
- Linea.

The project had 1 test, which passed successfully. The code coverage was 56.15%.

The total lines of code (LOC) of audited sources was 553.

Codebase update #1

After the initial audit, the codebase was updated. For the recheck, we were provided with commit 460cc7260a5c72a5f3eae6cfb95a0d05436179db.

Two medium severity issues were fixed. 54 out of 55 tests in **LeeaVault_full.t.sol** passed. The code coverage was 86.78%.

AUDIT PROCESS

The audit was conducted from October 30 to November 3, 2025.

We began by reviewing the provided materials, then held a call with the development team, during which they gave us a detailed project overview.

We performed a comprehensive manual review of all contracts within the audit scope, verifying the logic and security properties, including:

- Standard Solidity issues;
- The extent of the admin's influence over the project;
- The logic of profit calculation;
- The correctness of the share pricing mechanism;
- Whether the vault is protected against inflation attacks.

We analyzed the project with the following automated tools:

- AI scanner AuditAgent;
- AI scanner Savant Chat;
- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules;
- Semgrep rules for smart contracts.

We executed the test suite and calculated code coverage.

All identified issues — whether detected manually or through automated tools — were compiled into the private report.

The recheck was performed from November 7 to November 10, 2025. We verified the fixes, re-ran the tests, and updated the report.

MANUAL ANALYSIS

All contracts were manually inspected to validate their logic and security. Additionally, automated analysis results were manually verified. Confirmed issues are documented below.

Critical issues

Critical issues pose severe risks to project security, potentially leading to fund loss or other catastrophic failures. Contracts should not be deployed until such issues are resolved.

No critical issues were identified during the audit.

Medium severity issues

Medium severity issues may impact the project's current functionality. This category includes bugs, potential revenue loss, operational inefficiencies, and risks related to improper system management. We strongly recommend addressing these issues.

M01. Deposit/withdrawal imbalance at zero TVL (fixed)

Under normal conditions, consecutive deposit and withdrawal operations should result in a net zero profit or loss for a user (assuming no changes in TVL between the operations). However, this is not the case when deposits are made while the TVL is zero. In such scenarios, the user receives shares at a 1:1 ratio but is unable to fully withdraw their deposit due to the presence of dead shares (and potentially other users' shares).

> The special zero TVL case has been removed from the `proceedDeposit` function.

M02. Incorrect price per share calculation at zero TVL (fixed)

The `proceedDeposit` function updates the user's average price per share through the internal `_updateAverageEntryPps` function. However, it incorrectly processes deposits when the TVL is zero, effectively ignoring the value of the deposited assets. This issue results in an inaccurate performance fee calculation and often leads to reduced withdrawal amounts for the user.

> The special zero TVL case has been removed from the `proceedDeposit` function.

M03. Centralization risks (commented)

The current implementation relies heavily on privileged roles and off-chain modules. As a result, there are scenarios that could lead to undesirable outcomes for the project and its users.

- An off-chain module with the `PROCESSOR_ROLE` automatically processes deposit and withdrawal requests and also provides TVL data that influences the share price.
- The address holding the `DEFAULT_ADMIN_ROLE` is granted extensive privileges, including the ability to process large withdrawals, cancel deposit and withdrawal requests, execute arbitrary calls via the `multicall` function, modify multiple protocol settings, and upgrade the protocol.

Overall, this high degree of centralization introduces multiple risks that could negatively impact the project and its users, including potential denial-of-service conditions and partial or full loss of funds.

We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

> `Comment from the developers:`
> **Acknowledged. In the production architecture, the PROCESSOR_ROLE role will be owned by a multisig, which will verify the presence of signatures from offchain nodes responsible for calculating TVL offchain.**

M04. Insufficient code coverage (fixed)

The project has one test and insufficient code coverage. The existing test only verify basic logic and do not cover all critical scenarios.

We highly recommend expanding the test suite, ensuring all tests pass, and achieving sufficient code coverage to reliably validate the contract's behavior.

> **The developers have significantly expanded the test suite. However, one of the 55 relevant tests still fails, and the code coverage configuration remains non-functional.**

Low severity issues

Low severity issues do not immediately affect project operations but may introduce risks in future iterations. We recommend resolving these issues or providing justification for the chosen implementation.

L01. Redundant check

The code has several unnecessary checks:

- The check at line 162 in the `LeeaVault.deposit` function is redundant. For users submitting valid transactions, it causes extra gas consumption. Consider enforcing a non-zero value in setMinDepositAmount instead of checking in deposit, to reduce gas costs for regular users.
- The check `supplyBefore > 0` in the `LeeaVault.proceedRedeem` function is redundant. supplyBefore is always greater than 0 because dead shares are minted during initialize.
- The checks `sharesLocked > 0 && sharesLocked <= supplyBefore` are unnecessary:
    - `sharesLocked` cannot exceed `supplyBefore`;
    - `sharesLocked` is always greater than 0 because the redeem request creation already enforces `shares > 0` (line 315).

L02. Setters without events

The `setTreasuryAddress`, `setTraderSCAddress`, `setMinDepositAmount`, `setMinWithdrawAmount`, `setWithdrawTimeLimit` functions of the **LeeaVault** contract do not emit events. We recommend adding events to all functions that modify protocol settings to improve transparency and facilitate off-chain monitoring.

Known Issues

Prior to the audit, the developers provided the list of known issues present in the codebase. Below, we highlight the most significant ones and include our comments to further clarify their impact, associated risks, and potential solutions.

Cooldown limit

The `LeeaVault.proceedRedeem` function has a `withdrawTimeLimit` cooldown to prevent certain race conditions. This may cause legitimate, rapid back-to-back processing attempts by the off-chain service to fail, requiring a programmed delay.

> **Since only accounts with the `PROCESSOR_ROLE` can call the `proceedRedeem` function, race conditions can already be mitigated through controlled transaction submission and monitoring. We therefore recommend that users ensure the cooldown duration is reasonably small and that the risk of it interfering with legitimate withdrawal processing remains acceptable.**

Missed slippage protection

The final asset amount a user receives is calculated at the time of processing (`LeeaVault.proceedRedeem` function), not at the time of the request (`LeeaVault.redeem` function). This exposes the user to potential negative slippage if the asset price falls between their request and its finalization. While the user's entry PPS is snapshotted for fee calculation, the redeem function does not snapshot the vault's current TVL to lock in the user's withdrawal value. This is the root cause of the missed `minOut` slippage risk.

> **We acknowledge the complexity of managing large withdrawals that require manual processing and may fail the `minOut` check. However, we see no substantial technical barriers to implementing slippage checks for deposits and smaller withdrawals.**

Vault logic can be locked

A malicious actor can lock all **LeeaVault** operations by submitting a deposit or withdrawal request, which sets the vaultStatus to a pending state. The vault remains locked until an administrator manually cancels the pending request.

> **Even under normal conditions, the vault becomes temporarily locked, preventing all users from interacting with it. We see no significant technical barriers to implementing a non-blocking mechanism to mitigate this limitation.**

Front-running of initialize

The `LeeaVault.initialize` function is `public` and can be called by anyone on a newly deployed implementation contract. An attacker could front-run the legitimate owner's initialization transaction to seize control of the vault proxy.

> Consider deploying the implementation and calling initialize in a single transaction, for example through a Factory contract.

This analysis was performed by Pessimistic:

Daria Korepanova, Senior Security Engineer
Yhtyyar Sahatov, Security Engineer
Irina Vikhareva, Project Manager
Alexander Seleznev, CEO

November 11, 2025