Audit Report

Levana Stage 1

v1.0

January 13, 2022

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of this Report

Oak Security has been engaged by Levana to perform a security audit of the Levana protocol smart contracts.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following GitHub repository:

https://github.com/Levana-Protocol/levana-contracts/

Only the following directories of the repository have been audited:

- `contracts/levana-lli-farming`
- `contracts/levana-factory`
- `contracts/levana-staking`
- `contracts/levana-collector`
- `contracts/levana-treasury`
- `packages (except gov/balancer related files)`

Commit hash: `be201563f967639823714ab860c5048f856747a4`

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Levana allows for the creation of fungible tokens which represent exposure to leveraged assets. The protocol is implemented via a set of smart contracts that include LLI token creation and management, re-balancing, farming, staking and governance.

How to read this Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged** or **Resolved**. Informational notes do not have a status, since we consider them optional recommendations.

Note, that audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note, that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Summary of Findings

No	Description	Severity	Status
1	Owner is single point of failure of LLI farming contract, could cause funds to become inaccessible	Major	Resolved
2	Users depositing many small amounts may cause the LLI farming contracts' finalize farming, mint LLI and redeem rewards calls to run out of gas, leading to inaccessible deposits	Major	Resolved
3	Tiers of LLI farming contract can receive more deposits than specified in config	Major	Resolved
4	Updating the LLI farming contract's accepted asset could lead to inaccessible deposits and users not receiving their LLI tokens	Minor	Resolved
5	Finalize farming can be called even after first rewards have been redeemed, which implies that redeemed rewards were too high	Minor	Resolved
6	Updating the staking contract's reward asset will lead to inconsistent state and users losing/gaining value	Minor	Resolved
7	Missing validation of factory contract's distribution schedule could cause panics	Minor	Acknowledged
8	LLI farming contract allows owner to finalize farming and mint LLI tokens in the same block where last deposits arrive	Minor	Resolved
9	Updates to egg multipliers in the LLI farming contract can lead to too small or too big rewards being redeemed	Minor	Resolved
10	Rewards on deposits that have not been finalized can be redeemed, leading to too high rewards being paid	Minor	Resolved
11	Missing tax deduction in collector's swap, LLI farming's mint and the treasury's spend function will drain contracts' funds	Minor	Resolved
12	Adding many tokens to the factory contract could	Minor	Acknowledged

	cause distribution to run out of gas and makes queries more expensive		
13	Updating config values of the LLI farming contract may lead to inconsistent state	Minor	Resolved
14	Missing validation in collector's update config function allows high daily streaming fee values	Informational	Resolved
15	LLI farming contract allows CW20 tokens as the accepted asset, but deposit function only supports native tokens	Informational	Resolved
16	LLI farming contract's max total deposit config value is redundant	Informational	Resolved
17	Collector contract relies on invariant in balancer contract, which might inhibit future upgrades of the contracts	Informational	Resolved
18	Canonical address transformations are inefficient	Informational	Acknowledged
19	Unused reference to external contract in collector contract	Informational	Resolved
20	Hardcoded values should be constants	Informational	Resolved
21	Overflow checks not set for release profile in most packages	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium-High	-
Level of Documentation	Medium	-
Test Coverage	Medium-High	-

Detailed Findings

1. Owner is single point of failure of LLI farming contract, could cause funds to become inaccessible

Severity: Major

The current design of the LLI farming contract restricts the `GrabEggs`, `FinalizeFarming`, `MintLLI`, and `RedeemRewards` calls to the owner in `contracts/levana-lli-farming/src/contract.rs:210`, `248`, `312`, and `375`. That creates a single point of failure. Since there is no way for users to withdraw deposited tokens, an inaccessible or compromised owner account could render users' funds inaccessible.

Recommendation

We recommend removing the access control from the LLI token distribution contract. This can be achieved by tracking the total number of deposits made, eggs for deposits collected, deposits finalized, and deposits minted, and only allowing the next stage to be made once the numbers from the last stage match.

Status: Resolved

The Levana team changed the architecture such that grabbing eggs, finalization and minting is permissionless. Reward redemption still requires owner permissions – we still consider this issue as resolved since the principal can now be redeemed independent of the contract owner.

2. Users depositing many small amounts may cause the LLI farming contracts' finalize farming, mint LLI and redeem rewards calls to run out of gas, leading to inaccessible deposits

Severity: Major

The `finalize_farming` and `mint_lli` and `calculate_rewards` functions of the LLI farming contract iterate over an unbounded number of deposits in `contracts/levana-lli-farming/src/contract.rs:261`, `331` and `503`. Since the number of deposits per account can grow indefinitely, a user could add many entries, making the data structure grow too much to complete in a transaction due to gas limits, essentially blocking the operation.

This issue is signified by the fact that users can deposit zero tokens.

We do not consider this issue to be critical since the owner can use the pagination to skip accounts with too many deposits. At the same time, unsuspecting users might accidentally run into this issue, finding their funds frozen with no way to withdraw their deposit or redeem any

rewards. This could happen for example if bots or smart contracts interact with Levana and deposit many small amounts.

Recommendation

We recommend limiting the number of deposits per account, or aggregating deposits per tier to only have one deposit entry per tier. We also recommend setting a minimum deposit amount to prevent small entries.

Status: Resolved

3. Tiers of LLI farming contract can receive more deposits than specified in config

Severity: Major

The `deposit` function of the LLI farming contract checks whether the current tier has enough available space for the deposited amount in `contracts/levana-lli-farming/src/contract.rs:116`. If not, the left-over amount will be deposited into the next tier in line `123`, but without any validation, whether that amount will be bigger than the space available in the next tier.

In such a case of deposits that exhaust more the available space in two tiers, the tier deposit limits will be broken. That implies that higher rewards will be distributed than set in the config.

Recommendation

We recommend executing the tier space check in a loop until the deposit is fully assigned to tiers. Alternatively, we recommend asserting in the `instantiate` and `update_config` functions that every tier amount is greater than or equal to `max_deposit`. That would enforce that every deposit is smaller than or equal to the size of the smallest tier such that at maximum one tier can be filled.

Status: Resolved

4. Updating the LLI farming contract's accepted asset could lead to inaccessible deposits and users not receiving their LLI tokens

Severity: Minor

The LLI farming contract's `update_config` function allows the `accepted_asset` to be updated in `contracts/levana-lli-farming/src/contract.rs:193`. Such an update would leave previously deposited assets stuck in the contract and would lead to users getting LLI tokens minted for deposits in a wrong denom through the paged `mint_lli` function. Eventually, `mint_lli` would be unable to send the funds to the balancer contract

for minting LLI tokens in line 353, leading to some users neither receiving minted LLI tokens nor being able to retrieve their original deposits.

We classify this issue as minor, since it can only be caused by the contract owner.

Recommendation

We recommend removing the ability to update the `accepted_asset` once deposits have been made.

Status: Resolved

5. Finalize farming can be called even after first rewards have been redeemed, which implies that redeemed rewards were too high

Severity: Minor

The `finalize_farming` function of the LLI farming contract accepts a `finalize` argument that will lead the `FINALIZE_FARMING` storage item to be set to `true` in `contracts/levana-lli-farming/src/contract.rs:292`. Once it is set to `true`, calls to the `redeem_rewards` function are allowed. Redeeming rewards calls `calculate_rewards`, which reads the `TOTAL_MULTIPLIED_AMOUNT` storage item in line 497, and uses it to calculate the share of rewards a user will receive.

The owner can however still call the `finalize_farming` function, which could add to the `TOTAL_MULTIPLIED_AMOUNT` storage item through the call in line 289. Any users that redeemed rewards before would have received a too high share of the total rewards.

We classify this issue as minor, since it can only be caused by the contract owner.

Recommendation

We recommend adding an assertion to the `finalize_farming` function to ensure that the `FINALIZE_FARMING` storage item is `false`.

Status: Resolved

6. Updating the staking contract's reward asset will lead to inconsistent state and users losing/gaining value

Severity: Minor

The staking contract's `update_config` function allows the `reward_token` to be updated in `contracts/levana-staking/src/contract.rs:312`. Such an update would lead to existing rewards being claimed in a different token, which means that users may lose/gain

value, depending on the exchange rate between the previous and new reward token. Additionally, the contract's state will be inconsistent.

We classify this issue as minor, since it can only be caused by the contract owner.

Recommendation

We recommend removing the ability to update the `reward_token` if the total rewards in the contract are not zero.

Status: Resolved

7. Missing validation of factory contract's distribution schedule could cause panics

Severity: Minor

The factory contract's `distribution_schedule` is currently not validated in the `instantiate` and `update_config` functions in `contracts/levana-factory/src/contract.rs:56` and `183`. If the start date of the distribution schedule is greater than the end date, the subtraction in line `362` will panic.

Furthermore, the current implementation allows gaps and overlaps between distribution schedules, which might be intended.

We classify this issue as minor, since it can only be caused by the contract owner.

Recommendation

We recommend adding validation for the `distribution_schedule` to both the `instantiate` and the `update_config` functions.

Status: Acknowledged

8. LLI farming contract allows owner to finalize farming and mint LLI tokens in the same block where last deposits arrive

Severity: Minor

In the current implementation, deposits can arrive in the same block in which the owner finalizes farming or mints LLI tokens. That is caused by the fact that these owner actions can happen in or after the block configured as `farming_end` through the conditions in `contracts/levana-lli-farming/src/contract.rs:252` and `316`, while users can deposit tokens before or in the block configured as the `farming_end` through the condition in `contracts/levana-lli-farming/src/contract.rs:102`. Because of this overlap, deposits could be skipped.

Since skipped deposits can still be processed, we only consider this as a minor issue.

Recommendation

We recommend changing the conditions in `contracts/levana-lli-farming/src/contract.rs:214`, `252`, `316`, `379` and `451` from `env.block.time.seconds() < config.farming_end` to `env.block.time.seconds() <= config.farming_end` to allow finalization of farming and minting of LLI tokens only after the configured `farming_end`.

Status: Resolved

9. Updates to egg multipliers in the LLI farming contract can lead to too small or too big rewards being redeemed

Severity: Minor

The `grab_eggs` function can be called by the owner at any time, allowing updates to the stored `EGGS_MULTIPLIERS` map. During the `finalize_farming` function in `contracts/levana-lli-farming/src/contract.rs:262`, these egg multipliers are read from storage to compute the total multiplied amount and store it in the `TOTAL_MULTIPLIED_AMOUNT` item. In a separate call, during `redeem_rewards` in line `501`, the egg multipliers and the total multiplied amount are read again to determine the proportion of rewards for the redeemer. If an egg multiplier got updated between these two calls, that proportion would be off, leading to a too small or too big reward proportion being redeemed.

Recommendation

We recommend enforcing the sequence of the following messages: `GrabEggs` must be finished before `FinalizeFarming`, which must be finished before `MintLLI` and `RedeemRewards`.

Status: Resolved

10. Rewards on deposits that have not been finalized can be redeemed, leading to too high rewards being paid

Severity: Minor

During the paged `finalize_farming` function, an iteration is performed over all deposits in a page, which computes the `total_multiplied_amount` in `contracts/levana-lli-farming/src/contract.rs:273`. Every deposit considered during that calculation will get its `reward_finalized` field set to `true`. During reward calculation, the proportion of rewards a depositor receives is calculated by the proportion of deposits to that `total_multiplied_amount` in line `505`. Since there is no

guarantee that every deposit was actually processed by the paged `finalize_farming` function, it could be that some deposits were not considered during the calculation of the `total_multiplied_amount`. That would imply that the total rewards distributed can be bigger than the configured `total_rewards`.

Recommendation

We recommend requiring a deposit's `reward_finalized` field to be set to `true` in order to redeem rewards for that deposit.

Status: Resolved

11. Missing tax deduction in collector's swap, LLI farming's mint and the treasury's spend function will drain contracts' funds

Severity: Minor

The collector contract's `swap` function can deal with native and cw20 tokens. However, in the case of a native token, the code does not take into account taxes in `contracts/levana-collector/src/contract.rs:285`.

Similarly, the LLI farming contract's `mint_lli` function does not deduct taxes in `contracts/levana-lli-farming/src/contract.rs:353`.

Likewise, taxes are not accounted for in the treasury contract's `spend` function in `contracts/levana-treasury/src/contract.rs:62`.

Failure to deduct taxes will mean that the contracts' funds are slowly drained.

Recommendation

We recommend deducting taxes from the sent funds.

Status: Resolved

12. Adding many tokens to the factory contract could cause distribution to run out of gas and makes queries more expensive

Severity: Minor

The factory contract's `distribute` function iterates over all weights in `contracts/levana-factory/src/contract.rs:373`, which is unbounded. In the case of this growing to large (too many assets), a gas limit may be hit, leading to a revert of the call. Even without hitting limits, the gas cost of the `distribute` function grows with the number of tokens instantiated through the factory. The same issue exists for the `DistributionInfo` query in line `590`.

Similarly, the `query_addresses` function contains an unbounded iteration over all stored `addresses` in line `601`, making the query more expensive with more assets in the contract.

Recommendation

We recommend adding pagination to process a certain number of tokens/addresses at a time.

Status: Acknowledged

The Levana team intends to add no more than a few dozen tokens and monitor gas usage to not run into this issue.

13. Updating config values of the LLI farming contract may lead to inconsistent state

Severity: Minor

The LLI farming contract allows updates to various config values in `contracts/levana-lli-farming/src/contract.rs:162`. Those updates may lead to inconsistent state if they are performed after the `farming_end`:

- If `farming_end` is set to a later date after farming has been finalized or minting LLI has been started, deposits may be enabled again, leading to inconsistent state.
- If `total_rewards` or `tiers` is updated after some rewards have been redeemed already, there will be an inconsistency between the height of past and future reward redemptions.
- If `tiers` are updated after some rewards have been finalized, the total multiplied amount will be wrong.

Since only the owner can update these values, we only classify this issue as minor.

Recommendation

We recommend disabling updates to the `farming_end`, `total_rewards` and `tiers` values after the `farming_end`.

Status: Resolved

14. Missing validation in collector's update config function allows high daily streaming fee values

Severity: Informational

The `update_config` function of the collector contract allows the `daily_streaming_fee` parameter to be set to very large values in

`contracts/levana-collector/src/contract.rs:81`, which could destabilize the price of the underlying asset.

Recommendation

We recommend placing bounds on possible values for `daily_streaming_fee` to limit the power of the owner role and build trust among users.

Status: Resolved

15. LLI farming contract allows CW20 tokens as the accepted asset, but deposit function only supports native tokens

Severity: Informational

The LLI farming contract allows native and CW20 tokens as the `accepted_asset` in `contracts/levana-lli-farming/src/state.rs:29`, but the asset validation in the `deposit` function in `contracts/levana-lli-farming/src/contract.rs:106` only allows native tokens.

Recommendation

We recommend only allowing native tokens in the LLI farming contract's config.

Status: Acknowledged

16. LLI farming contract's max total deposit config value is redundant

Severity: Informational

The LLI farming contract's config has a `max_total_deposit` value in `contracts/levana-lli-farming/src/state.rs:26`. That value is redundant since an upper limit is implicitly applied through the sum of all tier amounts.

Recommendation

We recommend removing the `max_total_deposit` value to remove room for inconsistencies.

Status: Resolved

17. Collector contract relies on invariant in balancer contract, which might inhibit future upgrades of the contracts

Severity: Informational

During the collector contract's `Collect` message processing, the streaming fee is requested from the balancer contract. The expected fee is stored in `contracts/levana-collector/src/contract.rs:130`, and then it is used in the sub-message reply in line `166` for further processing. There is no validation though that the balancer contract actually sent the expected amount to the collector contract. If the amount is smaller than expected, funds of the collector may be used during the token swap in line `184`.

The current architecture makes the collector contract dependent on an invariant in the balancer contract, which might inhibit future upgrades of the contracts.

Recommendation

We recommend storing the collector's balance before requesting the streaming fee and then querying the current balance in the reply to calculate the received fee. That makes the architecture less entangled and allows for a more modular design, e. g. the balancer can apply a factor on the fee in the future.

Status: Resolved

18. Canonical address transformations are inefficient

Severity: Informational

While previously recommended as a best practice, usage of canonical addresses for storage is no longer encouraged. The background is that canonical addresses are no longer stored in a canonical format, so the transformation just adds overhead without much benefit. Additionally, the codebase is more complicated with address transformations.

Recommendation

We recommend removing any transformation from human to canonical addresses and vice versa.

Status: Acknowledged

19. Unused reference to external contract in collector contract

Severity: Informational

The collector contract is instantiated with a reference to the factory contract in `contracts/levana-collector/src/contract.rs:36`. However, this reference is not used in the contract and is unnecessary.

Recommendation

We recommend removing unused contract references.

Status: Resolved

20. Hardcoded values should be constants

Severity: Informational

The factory contract uses hardcoded values as parameters in `contracts/levana-factory/src/contract.rs:282-284`.

Recommendation

We recommend replacing magic numbers with constants for improved maintainability.

Status: Resolved

21. Overflow checks not set for release profile in most packages

Severity: Informational

While set implicitly through the workspace `Cargo.toml`, other packages do not explicitly enable overflow checks for the release profile. A future refactor may break implicitly enabled overflow checks, which could lead to security issues through undetected under- or overflows.

The following manifest files are affected:

- `contracts/levana-balancer/Cargo.toml`
- `contracts/levana-collector/Cargo.toml`
- `contracts/levana-factory/Cargo.toml`
- `contracts/levana-gov/Cargo.toml`
- `contracts/levana-lli-farming/Cargo.toml`
- `contracts/levana-staking/Cargo.toml`
- `contracts/levana-treasury/Cargo.toml`
- `packages/levana-protocol/Cargo.toml`
- `packages/mars/Cargo.toml`

Recommendation

We recommend enabling overflow checks in every package, even if no calculations are currently performed in the package, to prevent any issues when the code is extended or refactored in the future.

Status: Resolved