Audit Report

Levana Stage 2

v1.0

May 2, 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-balancer`
- `contracts/levana-gov`
- `contracts/levana-vesting`
- `packages (only files imported from balancer, gov and vesting contracts)`

Commit hash: `edebd1f61036fc6828e153f4ce3b58a06335b77d`

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 "LLI" tokens that 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, governance, and vesting.

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	Vesting contract allows unlimited allocation by malicious CW20 contracts	Critical	Resolved
2	Incorrect underlying assets could be deposited during minting	Critical	Resolved
3	Streaming fees cannot be collected	Critical	Resolved
4	Unvested tokens not refunded on termination	Critical	Resolved
5	Index parameters can be overwritten by re-registering the token	Minor	Resolved
6	Missing validation of rebalance info could lead to incorrect execution	Minor	Resolved
7	Lack of validation of vesting allocation	Minor	Resolved
8	Get asset price function called multiple times, which is inefficient	Informational	Resolved
9	Sub-message reply ids are unused	Informational	Resolved
10	Overflow checks not enabled in release profile	Informational	Resolved
11	Duplicate verification checks are inefficient	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	**High**	-
Code readability and clarity	**Medium-High**	-
Level of Documentation	**Low-Medium**	Documentation of high-level design and architecture is present. However, specific details of the complex leveraging mechanisms could be added.
Test Coverage	**Low-Medium**	-

Detailed Findings

1. Vesting contract allows unlimited allocation by malicious CW20 contracts

Severity: Critical

The CW20 receive hook of the vesting contract currently accepts any CW20 token. Therefore, through `contracts/levana-vesting/src/contract.rs:74-95`, it is possible for an attacker to create unlimited allocations using a malicious CW20 token contract that sets `cw20_msg.sender` to the protocol's admin address. The attacker could subsequently withdraw Levana tokens once the vesting period has expired.

Recommendation

We recommend adding a whitelist of trusted CW20 tokens and restricting the CW20 receive hook to those tokens.

Status: Resolved

Resolved in `d280bd2`

2. Incorrect underlying assets could be deposited during minting

Severity: Critical

When depositing into an index to mint new LLI tokens in file `contracts/levana-balancer/src/contract.rs:86-96`, a user could transfer the incorrect underlying assets for a specific index. In the case that the deposit occurs when no borrowing is required or the amount to borrow is less than the minimum rebalance amount the deposit could be accepted and the user issued with LLI tokens.

The consequence of this could be that the index is not able to correctly balance itself when required and could have its position liquidated. This is in addition to the issuance of LLI tokens without sufficient underlying deposits.

Recommendation

We recommend that the balancer contract verifies that deposits made through the CW20 hooks are made by the `lending_asset` specified in the `IndexInfo` struct of the index token to which the asset is being deposited.

Status: Resolved

Resolved in `d280bd2`

3. Streaming fees cannot be collected

Severity: Critical

During instantiation of the balancer contract the streaming fee collector contract is not set in the contract's config. Further, it is not possible to update the collector contract value using the `update_config` function.

This means that it will not be possible to collect the streaming fees as the function `collect_streaming_fee` will throw an error in `contracts/levana-balancer/src/contract.rs:245-251` if the collector contract is not defined in the config.

Recommendation

We recommend that the collector contract be defined during instantiation of the balancer contract.

Status: Resolved

Resolved in `bf822af`

4. Unvested tokens not refunded on termination

Severity: Critical

When terminating an allocation of Levana tokens in the vesting contract, the protocol calculates the amount vested until the current block timestamp, updates the allocation, and refunds the remaining tokens.

The amount to refund is calculated as the remainder of the total allocation amount minus the current amount vested. However, in `contracts/levana-vesting/src/contract:115` prior to calculating the amount to refund the allocation amount is set to equal the amount vested. This implies that the amount to refund is zero permanently locking the remaining tokens in the vesting contract.

Recommendation

We recommend that calculation of the amount to refund, `contracts/levana-vesting/src/contract:120`, is performed prior to the update of the allocation amount to that of the amount vested and the update of the allocation in storage in lines `115-116`.

Status: Resolved

Resolved in `d280bd2`

5. Index parameters can be overwritten by re-registering the token

Severity: Minor

During registration of LLI tokens in `contracts/levana-balancer/src/contract.rs:118` the parameters of each index token are defined. A subset of parameters of an index token can be updated using the `update_index_info` function. However, all parameters of an index token could be overwritten simply through re-registering an index token using the `register_index_token` function. This is problematic since this could lead to an inadvertent overwriting of an index token instance.

Recommendation

We recommend that the function `register_index_token` verifies that an index token of the same address has not been registered with the contract previously and returns an error.

Status: Resolved

Resolved in `d280bd2`

6. Missing validation of rebalance info could lead to incorrect execution

Severity: Minor

During registration of LLI tokens in `contracts/levana-balancer/src/contract.rs:118` the parameters of each index token are defined. However, the parameters of `RebalanceInfo` are not validated. This could lead to incorrect execution of the rebalance contract. For example, if the value of `min_ratio` was greater than `max_ratio`, leveraging down could potentially occur instead of leveraging up. Similarly, the balancer will not be able to leverage LLI tokens if `max_leverage_iterations` is not greater or equal to one.

Recommendation

We recommend validating the `RebalanceInfo` during the registration of LLI tokens.

Status: Resolved

Resolved in `d280bd2`

7. Lack of validation of vesting allocation

Severity: Minor

During the creation of an allocation of tokens in the vesting contract, `contracts/levana-vesting/src/contract.rs:92`, there is no validation of the `Allocation` struct. This could lead to a user being able to withdraw tokens prematurely or withdraw an incorrect amount.

For example, if `vesting_start` is in the past the user may be able to withdraw tokens early as the elapsed time would be greater than anticipated. Similarly, if the `vesting_cliff` is greater than the duration, a user could only vest their entire allocation following the expiration of the cliff.

Recommendation

We recommend that in addition to validation of the `allocation.amount` the protocol also validate, that `allocation.withdrawn` is initialized to zero; that `allocation.vesting_start` is not in the past or initialize as the current block timestamp; and that the `allocation.vesting_duration` is greater than the `allocation.vesting_cliff`.

Status: Resolved

Resolved in [d280bd2]

8. Get asset price function called multiple times, which is inefficient

Severity: Informational

Throughout the balancer contract `get_asset_price` is called twice in a row when the position of an LLI token is calculated, as shown in the snippet below:

```
let (lent, borrowed) = get_lent_and_borrowed(
    &deps.as_ref(),
    config.money_market_protocol.to_string(),
    env,
    &index_info,
)?;

let (_, ust_to_underlying) =
     get_asset_price(&deps.as_ref(),
index_info.lending_asset.clone())?;

let total_underlying = get_total_underlying(lent, borrowed,
ust_to_underlying)?;
```

In each instance the protocol retrieves the position from the money market protocol then queries the price of the underlying. Using these values to return the lent and borrowed values in the underlying and USD respectively. However, to calculate the total underlying position the protocol queries the asset price a second time to be able to value the borrowed position in the underlying. The current implementation queries the same data twice when performing the calculation, increasing both computational resources required and the complexity of the logic.

This occurs in the following 8 locations in the codebase:

- `contracts/levana-balancer/src/burn.rs:50-60`
- `contracts/levana-balancer/src/burn.rs:165-175`
- `contracts/levana-balancer/src/contract.rs:382-393`
- `contracts/levana-balancer/src/leverage_down.rs:61-79`
- `contracts/levana-balancer/src/leverage_up.rs:60-80`
- `contracts/levana-balancer/src/mint.rs:119-127`
- `contracts/levana-balancer/src/mint.rs:193-200`
- `contracts/levana-balancer/src/rebalance.rs:61-65`

Recommendation

We recommend returning both values from `get_lent_and_borrowed` in USD and removing the `get_asset_price` query. The `get_asset_price` can be queried outside the function prior to calculation of the total underlying position of an LLI token.

Status: Resolved

Resolved in [3d68652]

9. Sub-message reply ids are unused

Severity: Informational

Throughout the balancer contract, sub-messages are used to progress the balancer and leverage state machines. Upon execution of a sub-message the function `reply` is called in `contracts/levana-balancer/src/contract.rs:34`. However, the function only verifies that the sub-message is not an error but does not differentiate different sub-message IDs prior to continuing transition of the state machine.

Recommendation

We recommend performing identification of the `msg.id`.

Status: Resolved

Resolved in [d280bd2]

10. Overflow checks not enabled for release profile

Severity: Informational

The packages `contracts/levana-vesting/Cargo.toml` does not enable `overflow-checks` for the release profile.

While enabled implicitly through the workspace manifest, a future refactoring might break this assumption.

Recommendation

We recommend enabling overflow checks in all packages, including those that do not currently perform calculations, to prevent unintended consequences if changes are added in future releases or during refactoring. Note that enabling overflow checks in packages other than the workspace manifest will lead to compiler warnings.

Status: Resolved

Resolved in `d280bd2`

11. Duplicate verification checks are inefficient

Severity: Informational

When receiving CW20 tokens for reward deposits, the verification of `config.levana_token != info.sender` is performed twice in `contracts/levana-gov/src/contract.rs:117-119` and in lines `140-142`. Performing the same verification twice is inefficient.

Recommendation

We recommend removing the unnecessary second verification of the `info.sender` on lines `contracts/levana-gov/src/contract.rs:140-142`.

Status: Resolved

Resolved in `d280bd2`