Audit Report

Levana Perpetual Swaps

v1.0

May 3, 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 Perpetual Swaps 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 repositories:

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

Commit hash: `b44eba884752767720aca1d04e67fc137b8a7c7f`

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

Commit hash: `5b1c2eeb2c633b486a19cfe401b2c32e461e723d`

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

The Levana Perpetual Swaps protocol allows leveraged trading of any asset without the need of a counterparty. Trades are always executable and can be held indefinitely without an expiration date, hence the term "perpetual".

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**.

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	Anyone can whitelist a new vAMM or overwrite an existing one	Critical	Partially resolved
2	Risk fund is not able to partially disburse creditors	Critical	Acknowledged
3	`saveSwapInstruction` can be executed by anyone with arbitrary values	Critical	Acknowledged
4	`assert_admin` might run out of gas and contract-specific admins might lead to inconsistency and misconfiguration risks	Major	Resolved
5	Risk fund is not aware of the fees sent from the vault	Major	Acknowledged
6	`is_vamm_addr` might run out of gas	Major	Acknowledged
7	Prices collected from oracle are inverted	Major	Resolved
8	`CollectFee` message execution might run out of gas	Major	Acknowledged
9	Risk fund can be drained completely due to unpriced vega risk, if highly volatile assets are listed or volatility conditions change	Major	Resolved
10	Extensive admin permissions go against best practice	Major	Resolved
11	`bigint` crate is affected by CVE-2020-35880	Minor	Resolved
12	Address provider does not validate addresses	Minor	Resolved
13	vAMM's configuration parameters are not validated	Minor	Resolved
14	Admins can insert unvalidated vAMM addresses in the vault	Minor	Acknowledged
15	Some possible values of `vbase_liquidity` can make vAMM unusable	Minor	Resolved
16	It's not possible to change the admin of the address provider contract	Minor	Resolved
17	It's not possible to add or remove admins in	Minor	Resolved

	Factory, Spot Price, vAMM and Risk Fund contracts		
18	Lack of validation of AddTranche config values	Minor	Acknowledged
19	Iterations over tranches might run out of gas	Minor	Acknowledged
20	Funds sent to the Vault with a denomination different from the expected one are lost	Informational	Resolved
21	Inefficiency in `OpenPosition` execution flow	Informational	Acknowledged
22	Inefficiency in `ClosePosition` execution flow	Informational	Resolved
23	Handling of duplicate vAMMs is inefficient	Informational	Resolved
24	Inefficiency during funding	Informational	Acknowledged
25	Inefficiency in withdrawals from vault	Informational	Resolved
26	Outdated cosmwasm related dependencies	Informational	Resolved
27	Development utilities should be removed from production code	Informational	Acknowledged
28	Unfinished error handling	Informational	Acknowledged

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium	-
Level of documentation	High	-
Test coverage	Medium-High	-

Detailed Findings

1. Anyone can whitelist a new vAMM or overwrite an existing one

Severity: Critical

In `contracts/factory/src/contract.rs:45`, when executing `ExecuteMsg::Whitelist` message, there is no check on the sender's address of the transaction to ensure that it is contained in the admins list.

This may lead to three malicious behaviors:

- Anyone can instantiate and whitelist a new vAMM on behalf of admins passing arbitrary parameters.
- As vAMMs are indexed in the `VAMM_ADDRS Map` with the stringification of `currency_pair` as the key, anyone can overwrite an existing vAMM address mapping with a new one with arbitrary parameters.
- Using the `DelTranche` in `contracts/risk-fund/src/contract.rs:86`, maliciously registered vAMMs can drain the risk fund completely.

Recommendation

We recommend restricting `ExecuteMsg::Whitelist` to be executed only from addresses included in the admins list.

We also recommend disabling the whitelist of new vAMMs that already exist.

Status: Partially resolved

This has been partially addressed, it is still possible to whitelist vAMMs that already exist.

2. Risk fund is not able to partially disburse creditors

Severity: Critical

In `contracts/risk-fund/src/state/tranche.rs:366`, when disbursing from the risk fund, the execution tries to take all possible funds from tranches in order to pay the creditor. If there aren't enough funds, it will return an error causing creditors to not get funds, not even partially based on the availability.

Recommendation

We recommend disbursing creditors to the maximum possible amount of funds available when it is partially insolvent.

Status: Acknowledged

3. `saveSwapInstruction` can be executed by anyone with arbitrary values

Severity: Critical

In `contracts/risk-fund/src/contract.rs:73`, there isn't any check on the message sender role when executing a `saveSwapInstruction` message. As this message is used to save and update the Astroport swap pair contract address for a specific pair, the execution of it should be restricted to admins only.

In fact, a malicious actor could overwrite a legit Astroport swap pair address to a malicious one. Such a malicious pair could receive UST, emit appropriate events, but not actually return another asset. This is also possible because `ReplyKind::AstroportSwapUst` parses the asset information from the returned event.

Recommendation

We recommend adding a condition that only allows admins to execute the `saveSwapInstruction` message.

Status: Acknowledged

4. `assert_admin` might run out of gas and contract-specific admins might lead to inconsistency and misconfiguration risks

Severity: Major

In `contracts/factory/src/state/admin.rs:13`, iterations over `ADMIN_ADDRS` may run out of gas if the vector has too many entries. Because admins cannot be removed there is currently no way to recover from this issue (see issue 11). Additionally, different contracts, e.g. the factory, the vAMMs, and the vault, might all contain distinct lists of admins, which introduces the risk of misconfiguration.

Recommendation

We recommend adding a maximum of admins and adding a function to remove admins. In addition, we recommend unifying/centralizing the admins in the address provider or implementing a logic that ensures consistency of admins across contracts.

Status: Resolved

5. Risk fund is not aware of the fees sent from the vault

Severity: Major

In `contracts/vamm/src/contract.rs:98`, `contracts/vamm/src/state/liquidation.rs:309` and `314`, the vAMM is sending a

message to the vault in order to send collected fees to the risk fund using `VammMsg::SendFunds`.

The execution of this message is sending funds to the risk fund through a `BankMsg::Send` but it's not directly triggering a mechanism to collect fees.

This may lead to the contract not being aware of the collected fees and to capital inefficiency of fees that are already in the risk fund but are not in tranches.

Recommendation

We recommend implementing a mechanism to automatically collect funds in the tranches when fees are sent from the vault to the risk fund.

Status: Acknowledged

6. `is_vamm_addr` might run out of gas

Severity: Major

In `contracts/factory/src/state/address.rs`, the `is_vamm_addr` function has an asymptotic cost of O(n) and can make the message run out of gas depending on the length of `VAMM_ADDRS` and the `to_check` parameter position in the array. This can happen when calling `QueryMsg::IsVamm` from the risk fund's `can_disburse` function.

Since any user can add vAMMs (see issue 1), an attacker can exploit this vulnerability to deny the expected functionality.

Recommendation

We recommend limiting the number of vAMMs, applying pagination, or introducing an O(1) approach to check if an address is a vAMM.

Status: Acknowledged

7. Prices collected from oracle are inverted

Severity: Major

During liquidation, the current price is calculated as `UST/asset`, in the same way as it is calculated for `spot_price` or `mark_price` in `contracts/vamm/src/state/market.rs:112-119`.

As prices returned from Chainlink are represented as `asset/UST` and they use a different method respecting `mark_price`, calculations between those two values are inconsistent.

Recommendation

We recommend inverting the spot prices returned from oracles in order to be consistent with other values represented as `UST/asset`.

Status: Resolved

8. `CollectFee` message execution might run out of gas

Severity: Major

In `contracts/risk-fund/src/contract.rs:62`, under certain circumstances, the execution of the `CollectFee` message can consume a big amount of gas and in the worst case can run out of gas.

Consider as an example the scenario that the UST tranche is almost full.

1. Alice sends a `CollectFee` message.
2. The execution will call the `collect_fee` function and will fill the UST vault with UST in the msg funds.
3. Now, assume that this will not consume all the funds in the msg. The execution will call recursively `collect_fee` in order to fill the second tranche.
4. This will cause an interaction with Astroport, we need 2 queries to correctly create the message and then send it.
5. Astroport contract execution returns a reply.
6. Contract uses the reply to update the tranche state.
7. If there still are some funds in the msg, the execution will call another time the `collect_fee` function.
8. Here we can potentially loop several times until the execution will fill all the tranches or user funds sent in the message will be depleted.

This issue is not unlikely to happen if the UST tranche is almost full.

Recommendation

We recommend reworking the `collect_fee` function in order to avoid using recursion, which can grow the stack unnecessarily, and adding a hard cap to how many tranches a user can fill in one interaction.

Status: Acknowledged

9. Risk fund can be drained completely due to unpriced vega risk, if highly volatile assets are listed or volatility conditions change

Severity: Major

Suppose that an asset faces above normal volatility conditions and suppose that an attacker creates two large and maximum leveraged positions with two distinct wallets one short (A) one long (B) such that the expected sum of the payout of the two positions is `A+B-fees`, and the expected value of `A+B` is `E(A+B) = 0`. If the market volatility is large enough to assure that the collateral of one position will not be enough to cover its losses, an exploiter can create a series of payments to drain all funds from the risk fund. In layman's terms this is because the losses of the losing position are limited (since bad debt doesn't exist) and the earnings of the winning position are not capped but are paid out entirely until the risk fund is empty.

Because such an attack requires sound financial engineering and is only possible during adverse market conditions of a given asset, we do not consider this to be a major issue.

However, this attack vector also highlights another important vulnerability of the protocol. Because very risky/volatile assets are covered by the same tranches of the risk fund as less risky assets, less risky positions might lose their risk fund coverage from adverse events in the more risky segment. Especially for risk averse users that use the protocol for hedging purposes, this might be an undesired property.

Recommendation

We recommend re-evaluating margin practices and pricing in the vega risk of the underlying asset. These could include requiring higher contributions to the risk fund from more risky assets or separating risk funds by asset classes. Additionally, limits to position size or delays of payouts when market volatility is high could be considered. Even in traditional finance, there is no consensus about optimal margin models (see https://www.bis.org/bcbs/publ/d526.pdf) but in all cases the margin shall increase with the volatility of the underlying - i.e. price in the vega risk. The SIMM (Standard Initial Margin Model for Non-Cleared Derivatives) has become an industry standard and might provide valuable insights for improving the collateral and margin framework.

Status: Resolved

10. Extensive admin permissions go against best practice

Severity: Major

It is best practice to restrict admin permissions to actions that do not directly allow access to or could create a loss of user funds. Under the current architecture, admins have several such permissions:

1. In `contracts/vamm/src/contract.rs:224`, using `ExecuteAdminMsg::ForceMarketPrice`, the market price can be set to arbitrary values putting users' funds at risk of liquidation.

2. In `contracts/vault/src/contract.rs:67`, any address can be added as a new vAMM's address using `ExecuteMsg::InsertVamm`, which then can drain funds from the vault completely via `VammMsg::SendFunds`.

3. In `contracts/vamm/src/state/config.rs:69-111`, the `config` can be updated such that:
    a. Only full liquidations take place, via updates of `config.liquidation_total_ratio`.
    b. The likelihood of liquidations increases drastically, via updates of `config.mark_price_divergence`.
    c. Liquidation rewards are sent fully to the liquidating address, via updates of `config.liquidation_reward_split`.
4. In `contracts/risk-fund/src/contract.rs:86`, `DelTranche` can be called by admins and vAMMs. It sends the funds to the sender.
5. In `contracts/risk-fund/src/contract.rs:35`, it is possible to add a new admin but not possibility to remove an admin. This makes it impossible to stop malicious admins.

This issue is especially problematic due to the smart contracts' support of multiple admin accounts – administering these requires off-chain maintenance, which is prone to human error.

Recommendation

We recommend either removing the `ExecuteAdminMsg::ForceMarketPrice` function or limiting its capabilities drastically. In addition, we recommend limiting the functionality of the `VammMsg::SendFunds`. In addition, related to issue 7, we recommend limiting and validating the parameter values to be within a range that protects users. Finally, we recommend either removing the `DelTranche` functionality completely or only allowing admins to call it. The funds should be transferred to another tranche, not to an admin address.

Status: Resolved

11. `bigint` crate is affected by CVE-2020-35880

Severity: Minor

In `rust/utils/Cargo.toml:17` of the `levana-common` repository, `bigint` is specified as a dependency. As reported in https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-35880 and https://rustsec.org/advisories/RUSTSEC-2020-0025.html that crate is affected by a Critical CVE with a score of 9.8.

The crate is not maintained anymore and contains several known bugs (including a soundness bug).

Recommendation

We follow the recommendation in the CVE to substitute `bigint` with https://crates.io/crates/uint.

Status: Resolved

12. Address provider does not validate addresses

Severity: Minor

In `contracts/address-provider/src/contract.rs:44`, `ExecuteMsg::Change` function handler `execute_change` takes a `to` parameter of type `Addr`. This may cause an unvalidated address to be stored.

Recommendation

We recommend changing the `to` parameter to be of type `string` and adding validation to obtain an `Addr` in `execute_change`.

Status: Resolved

13. vAMM's configuration parameters are not validated

Severity: Minor

In `contracts/vamm/src/state/config.rs:69`, the `update_config` function which is called when initializing and updating `config` lacks validation on configuration parameters before saving them.

As some of them are intended to be in a specific range and used in math operations, this may lead to inconsistency and execution errors caused by division by zero or assigning negative values to unsigned integers.

Recommendation

We recommend adding validation of configuration parameters. At a minimum, it is recommended to ensure that the values are greater or equal to zero and less than or equal to one.

Status: Resolved

14. Admins can insert unvalidated vAMM addresses in the vault

Severity: Minor

In `contracts/vault/src/contract.rs:70`, `insert_vamms` allows adding a new vAMM address in the `VAMM_ADDRS` array without validation.

This may cause inconsistencies in `VAMM_ADDRS`.

Recommendation

We recommend validating appropriately the new address before adding it to the list.

Status: Acknowledged

15. Some possible values of `vbase_liquidity` can make vAMM unusable

Severity: Minor

In `contracts/vamm/src/state/markets.rs:158`, `init_curve` is using `vbase_liquidity` taken from the `InstantiateMsg` to initialize `quote`, `base` and `k` of the curve.

As there is no validation for `vbase_liquidity`, a value of this parameter equals to or less than 0 can make the vAMM unusable as this will cause:

- Case `vbase_liquidity == 0`: `quote`, `base` and `k` all with zero value
- Case `vbase_liquidity < 0`: `quote` and `base` with a negative value

Recommendation

We recommend implementing validation on `vbase_liquidity` to make sure that is greater than zero.

Status: Resolved

16. It's not possible to change the admin of the address provider contract

Severity: Minor

In `contracts/address-provider/src/contract.rs`, the admin `cw_controller` is registering the contract admin address in the instantiate message. After that, it's not possible for the admin to change its address.

Recommendation

We recommend implementing a message that allows the admin to change its address. As the contract is using the admin `cw_controller`, it's possible to use the built-in `execute_update_admin` from https://docs.rs/cw-controllers/latest/cw_controllers/struct.Admin.html#method.execute_update_admin

Status: Resolved

17. It's not possible to add or remove admins in Factory, Spot Price, vAMM and Risk Fund contracts

Severity: Minor

In Factory, Spot Price, vAMM and Risk Fund contracts' `instantiate` function, the list of admins is taken from the `InstantiateMsg` message.

After the instantiation, it's not possible for admins to be added or removed. This is problematic since a compromised admin address cannot be removed or replaced with a new one.

Recommendation

We recommend implementing a message that allows admins, or a quorum of them, to add/remove new addresses. As mentioned in issue 2, we also recommended implementing logic that ensures consistency of admins across contracts.

Status: Resolved

18. Lack of validation of `AddTranche` config values

Severity: Minor

In `contracts/risk-fund/src/contract.rs:80` when executing the `AddTranche` message, values in the config object are not validated. This can lead to a not working tranche registered in the contract.

Recommendation

We recommend implementing validation of the new tranche configuration.

Status: Acknowledged

19. Iterations over tranches might run out of gas

Severity: Minor

In `contracts/risk-fund/src/state/tranche.rs:59-73`, iterations over the tranches might run out of gas, if too many tranches exist. As this can happen only through admin error and is recoverable, we classify this as a minor issue.

Recommendation

We recommend adding a limit to the number of tranches.

Status: Acknowledged

20. Funds sent to the Vault with a denomination different from the expected one are lost

Severity: Informational

In `contracts/valut/src/contract.rs:53`, a check is performed that ensures that in the transaction there is a `Coin` with the expected `denom` field.

This validation does not ensure that no other native tokens are sent though, and any additional native tokens are not returned to the user, so they will be stuck in the contract forever.

Recommendation

We recommend checking that the transaction contains only the expected `Coin` using the `must_pay` function, see https://docs.rs/cw0/0.9.1/cw0/fn.must_pay.html.

Status: Resolved

21. Inefficiency in `OpenPosition` execution flow

Severity: Informational

In `contracts/vamm/src/contract.rs:129`, the `OpenPosition` message handler has the following flow:

1. `ExecuteMsg::OpenPosition`
2. `VammMsg::Withdraw`
3. `ReplyId::OpenPosition`
4. `VammMsg::SendFunds`

As `ReplyId::OpenPosition` is not taking any input from the response, and as an error thrown from `VammMsg::Withdraw` will revert also `ExecuteMsg::OpenPosition` there's no need for `ReplyId:OpenPosition`.

Recommendation

We recommend merging the code of `ReplyId::OpenPosition` directly inside `ExecuteMsg::OpenPosition`. This will simplify the flow and also reduce the amount of spent gas.

Status: Acknowledged

22. Inefficiency in `ClosePosition` execution flow

Severity: Informational

In `contracts/vamm/src/contract.rs:133`, the `ClosePosition` message handler has the following flow:

1. `ExecuteMsg::ClosePosition`
2. `VammMsg::Deposit`
3. `ReplyId::ClosePosition`

As `ReplyId::ClosePosition` is not taking any input from the response, and as an error thrown from `VammMsg::Deposit` will also revert `ExecuteMsg::ClosePosition` there's no need for `ReplyId:ClosePosition`.

Recommendation

We recommend merging the code of `ReplyId::ClosePosition` directly inside `ExecuteMsg::ClosePosition` to reduce the complexity and the amount of gas spent.

Status: Resolved

23. Handling of duplicate vAMMs is inefficient

Severity: Informational

In three instances duplicate vAMMs might be handled, which is inefficient:

- `contracts/factory/src/contract.rs:75`
- `contracts/vault/src/state/admin.rs:49-59`
- `contracts/factory/src/state/reply.rs:39`

Recommendation

We recommend adding a check if the vAMM already exists and returning an error if it does.

Status: Resolved

24. Inefficiency during funding

Severity: Informational

In `contracts/vamm/src/state/funding.rs:41`, the function continues to execute even if there are no positions to handle, which is inefficient.

Recommendation

We recommend terminating execution with a return value if line `contracts/vamm/src/state/funding.rs:41` is reached.

Status: Acknowledged

25. Inefficiency in withdrawals from vault

Severity: Informational

In `contracts/vault/src/contract.rs:93-104` and `contracts/vault/src/state/account.rs:69`, zero amounts can be withdrawn, which is inefficient.

Recommendation

We recommend terminating execution if the withdrawal amount is zero.

Status: Resolved

26. Outdated cosmwasm related dependencies

Severity: Informational

In both `levana-common` and `levana-perpetual-swap-contracts`, crates are specifying in `Cargo.toml` an outdated version for various comswasm related dependencies such as: `cosmwasm-std`, `cw-storage-plus`, `cosmwasm-storage`, `cw-storage-plus`, `cw2`, etc.

Recommendation

We recommend updating those dependencies to version 0.9.1.

Status: Resolved

27. Development utilities should be removed from production code

Severity: Informational

In `contracts/vamm/src/contract.rs:107` and `contracts/vamm/src/contract.rs:154`, there are a couple of blocks of code used for development purposes that are skipped using the `cfg!` macro. That macro is evaluated at compile-time, but unlike the `#[cfg]` attribute, it does not remove the code.

Recommendation

We recommend using the `#[cfg]` attribute where possible.

Status: Acknowledged

28. Unfinished error handling

Severity: Informational

In `contracts/risk-fund/src/state/tranche.rs:251`, there an `Err` of `Err(ContractError::unimplemented("TODO"))` is returned.

Recommendation

We recommend completing the error handling.

Status: Acknowledged