Security Audit Report

Margined Protocol Locust Vault Framework

v1.0

November 25, 2024

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

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

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by the Neutron Audit Sponsorship Program to perform a security audit of Margined Protocol's Locust Vault Framework.

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 target:

Repository	https://github.com/margined-protocol/transfigure
Commit	`df3b088c5ac1ecce87793aeb9b15d83c4a1f71fe`
Scope	All contracts were in scope.
Fixes verified at commit	`67ab1b2a6e12e83f5b1e024d73ed02c10320863b` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

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

Margined Protocol Locust Vault Framework enables non-custodial executions of sophisticated trading strategies.

The fund contract implements a vault extension that accepts user deposits in return for shares, which can be redeemed along with trading profit. The strategy contract will withdraw the funds to perform various trading activities executed by an off-chain controller bot. Authorizations are granted to the bot via `authz` so it can trade on behalf of the strategy contract.

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**, **Partially Resolved**, or **Resolved**.

Note that audits are an important step to improving 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 in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium-High	-
Level of documentation	Medium-High	Documentation was available at https://docs.margined.io/vaults/overview and in the `README`.
Test coverage	Medium-High	`cargo-llvm-cov` reports the following test coverage: • `37.18%` function coverage • `63.66%` line coverage • `20.56%` region coverage

Summary of Findings

No	Description	Severity	Status
1	User redeemed amount can be manipulated to break future deposits	Critical	Resolved
2	Incorrect calculation of deposit value causes more shares to be minted	Critical	Resolved
3	Users might withdraw less liquidity when the strategy contract borrowed funds	Critical	Resolved
4	Controller can bypass the float threshold by specifying duplicates	Major	Resolved
5	Strategy contract's ownership cannot be transferred	Major	Resolved
6	`estimate_cycle_profit` configuration field cannot be updated	Major	Resolved
7	Users might receive zero shares due to small deposits, causing a loss of funds	Major	Resolved
8	Query messages do not account for `token1` denom	Minor	Resolved
9	Controller address cannot be updated	Minor	Resolved
10	`event_burn` is not called during redemption	Informational	Resolved
11	Unnecessary storage save operation	Informational	Resolved
12	Incorrect token denom emitted in events	Informational	Resolved
13	Unused functions and errors	Informational	Resolved
14	Unimplemented query messages	Informational	Resolved
15	Misleading event types and comments	Informational	Resolved

Detailed Findings

1. User redeemed amount can be manipulated to break future deposits

Severity: Critical

In `contracts/fund/src/handle.rs:206-217`, the `handle_redeem` function computes the `user_deposit_redeemed` variable as the withdrawn amount. The withdrawn amount is calculated based on two factors: the ratio of shares amount to burn divided by the total shares owned by the user and the state balance of the user deposit `UserDeposit.total_deposits`. After that, the withdrawn amount will be deducted from the user deposit state and the total staked tokens `State.total_staked_tokens`.

This approach is problematic because the withdrawal amount can be manipulated:

- The shares are native tokens minted from the `tokenfactory` module, which can be transacted like regular tokens. These tokens can be transferred to other recipients who do not have the same `UserDeposit.total_deposits` as the sender. This would cause the withdrawn amount to be incorrect as the recipient has a different `UserDeposit.total_deposits` value.
- The burn ratio is computed based on the current balance of the shares owned by the caller. Since the tokens can be transferred to other users, the burn ratio can be manipulated if the caller temporarily owns a large portion of the total supply. For example, this may be achievable via a flash loan.

The computed withdrawal amount is correlated with the strategy cap threshold, which enforces the maximum amount of funds that can be deposited. Using the above methods, attackers can manipulate the withdrawal amount to be less than intended, decreasing the total staked tokens by a little. This causes the total staked tokens to be artificially inflated, preventing future deposits from occurring due to the `check_strategy_cap` validation in `contracts/fund/src/handle.rs:104`.

Recommendation

We recommend introducing a `BlockBeforeSend` sudo hook that is executed every time the share tokens are transacted. The hook should update the sender and recipient's `UserDeposit.total_deposits` state based on the ratio of the transacted amount over the total shares held.

Additionally, the `tokenfactory` module's address will be the sender address when minting shares and the recipient address when burning shares. The hook should ignore updating the `UserDeposit` states in these scenarios, as the module does not hold the shares.

Status: Resolved

2. Incorrect calculation of deposit value causes more shares to be minted

Severity: Critical

In `contracts/fund/src/helpers.rs:72-79`, the `calculate_total_value` function queries the TWAP price via `StrategyQueryMsg::TwapPrice` and multiplies it with tokens in `token1` denom to compute the deposit value denominated in `token0`. Internally, an `ArithmeticTwapToNowRequest` query is dispatched to the Osmosis pool with the `base_asset` parameter as `token0` and the `quote_asset` parameter as `token1`, as seen in `contracts/strategy/src/query.rs:111`.

When the `base_asset` parameter is set to `token0`, the price returns in `token1/token0` format, representing how much `token1` is required per one `token0`. For example, if the base asset is OSMO and the quote asset is USDC, the query returns how much USDC is needed to purchase one OSMO.

The issue is that the `token1` tokens are incorrectly multiplied by the price when computing the deposit value in `token0` (see `contracts/fund/src/helpers.rs:78`). This causes the computed value to be larger than intended, resulting in more shares minted for the caller. An attacker can exploit this issue by depositing many `token1` tokens and withdrawing them immediately to steal funds from the contract, causing a loss of funds.

Recommendation

We recommend dividing `token1` tokens by the TWAP price when computing the deposit value.

Status: Resolved

3. Users might withdraw less liquidity when the strategy contract borrowed funds

Severity: Critical

In `contracts/fund/src/handle.rs:197-203`, the `handle_redeem` function computes the number of assets to return based on the current fund contract balance. This is problematic because the strategy contract can borrow funds anytime for trading purposes, causing the fund contract balance to be decreased temporarily. If users withdraw liquidity during these periods, they may not be able to withdraw the full amount of funds they initially deposited, causing a loss of funds.

Depending on the `float` threshold configuration, the strategy contract can, for example, only borrow up to 90% of the fund contract's balance. However, users with large positions, for example of 10% or more, cannot redeem their shares fully without experiencing a loss in the withdrawn liquidity.

Recommendation

We recommend implementing a `min_receive` parameter to ensure users do not receive fewer funds than intended when withdrawing liquidity.

Status: Resolved

4. Controller can bypass the float threshold by specifying duplicates

Severity: Major

In `contracts/fund/src/handle.rs:326-365`, the `handle_withdraw` function allows the controller to withdraw tokens from the contract with a restriction that it does not exceed the float threshold. For example, if the threshold is set to 10%, the controller can only borrow up to 90% of the contract balance.

However, the controller can specify the `tokens_to_withdraw` vector with duplicate coins to bypass the validation. This is because the `BankMsg::Send` messages in line `359` have not been dispatched yet, causing the `get_balance` function to exclude previous withdrawals, ultimately circumventing the withdrawal limit.

Recommendation

We recommend returning an error if the `tokens_to_withdraw` vector contains duplicates.

Status: Resolved

5. Strategy contract's ownership cannot be transferred

Severity: Major

In `contracts/strategy/src/contract.rs:36`, the `instantiate` function stores the admin address in the `Config.admin` state. The admin is a privileged address to update the vault address and `authz` grants via the `SetVault` and `SetGrants` messages.

The issue is that the strategy contract does not implement any entry points to update the admin address. This is problematic because if the current admin is compromised, it is impossible to update the admin to a different address.

Recommendation

We recommend implementing a two-step ownership transfer. The flow can be as follows:

1. The current owner proposes a new owner address that is validated with `addr_validate`.
2. The new owner account claims ownership, which applies the configuration changes.

Status: Resolved

6. `estimate_cycle_profit` configuration field cannot be updated

Severity: Major

In `contracts/fund/src/config.rs:74`, the `handle_update_config` function allows the contract owner to update configurations specified in the `UpdateConfig` struct, which includes the `estimate_cycle_profit` field. However, the field is ignored throughout the `handle_update_config` function, preventing the `Config.estimate_cycle_profit` value from being updated.

Consequently, the profit percentage will default to zero when the controller does not explicitly specify `cycle_profit` during repayment (see `contracts/fund/src/handle.rs:391`), potentially causing an incorrect profit computation.

Recommendation

We recommend modifying the `handle_update_config` function to update the `estimate_cycle_profit` field.

Status: Resolved

7. Users might receive zero shares due to small deposits, causing a loss of funds

Severity: Major

In `contracts/fund/src/handle.rs:131-156`, the `handle_deposit` function mints the shares to the user if the number of shares exceeds zero. This is problematic because the share amount can be zero if the user's deposited liquidity is insufficient to warrant a single share. For example, this may happen if the fund contract has received a lot of trading profit from the strategy contract.

The deposit transaction will succeed since `handle_deposit` only mints shares that exceed zero, resulting in the user's deposits remaining in the contract but receiving nothing in return. This causes a loss of funds for users who deposit tiny liquidity.

Recommendation

We recommend returning an error if the number of shares to mint is zero.

Status: Resolved

8. Query messages do not account for `token1` denom

Severity: Minor

The fund contract implements functionality to optionally support `token1` configuration along with `token0`, as seen in `contracts/fund/src/config.rs:18`. However, the `query_preview_deposit`, `query_total_assets`, and `query_convert_to_assets` functions do not account for `token1` deposits.

For example, the `query_preview_deposit` function in `contracts/fund/src/query.rs:27-41` does not include the `config.token1` contract balance when computing the number of shares to mint. This is incorrect because the response will be inaccurate compared to the actual deposit transaction, potentially misleading users and third parties.

Recommendation

We recommend reviewing and updating query messages to include the `token1` denom if configured.

Status: Resolved

9. Controller address cannot be updated

Severity: Minor

In `contracts/strategy/src/contract.rs:37-52`, the `instantiate` function grants `authz` authorizations to the controller address so it can trade on behalf of the strategy contract.

The issue is that the strategy contract does not implement entry points for the admin to update the controller address. This is problematic because if the controller is compromised, the admin cannot update the controller to a different address to limit the attack surface.

Recommendation

We recommend implementing an entry point for the admin to update the controller address. The `revoke_authz_grant_messages` and `create_authz_grant_messages` functions should be called to revoke grants from the previous controller and to grant authorizations for the new controller.

Status: Resolved

10. `event_burn` is not called during redemption

Severity: Informational

Token burning occurs when users receive underlying funds in exchange for liquidity tokens during the redemption phase. In `contracts/fund/src/events.rs:27`, the `event_burn` function is implemented but never called.

Consequently, if off-chain components monitor token-burning events, it may not work properly because the event is never emitted.

Recommendation

We recommend analyzing whether the `event_burn` function should be called during the `handle_redeem` phase. If not required, its implementation should be removed from the codebase.

Status: Resolved

11. Unnecessary storage save operation

Severity: Informational

In `contracts/fund/src/handle.rs:68`, the `handle_instantiate` function stores the `State` struct in the storage. This is unnecessary because there is no mutation towards the `state` variable after it is retrieved in line `49`, making it a redundant operation.

Recommendation

We recommend removing the `state.save_to_storage(&mut deps)?`.

Status: Resolved

12. Incorrect token denom emitted in events

Severity: Informational

In `contracts/fund/src/handle.rs:431`, the `handle_repay` function emits the `token_in` denom as `config.token0`. This is incorrect because the contract may be configured to support `config.token1`, causing the emitted token denom to be inaccurate.

Recommendation

We recommend setting the denom to emit `repayment.denom`.

Status: Resolved

13. Unused functions and errors

Severity: Informational

The `fund` and `strategy` contracts implement functions and custom errors that are not used anywhere in the code:

- `query_spot_price` in `contracts/fund/src/queries.rs:42`
- `update_state` in `contracts/fund/src/state.rs:61`
- `InvalidFunds`, `MigrationError`, and `NonPayable` errors in `contracts/strategy/src/errors.rs`.

The existence of such functions and objects violates best practices. They obfuscate the code and unnecessarily decrease the code's maintainability and readability.

Recommendation

We recommend analyzing whether these functions and errors should be used. If not, their implementation should be removed from the codebase.

Status: Resolved

14. Unimplemented query messages

Severity: Informational

The `ConvertToShares` query in `contracts/fund/src/contract.rs:138` is not implemented. This reduces the user experience and negatively affects the code's reliability.

Recommendation

We recommend implementing the missing query.

Status: Resolved

15. Misleading event types and comments

Severity: Informational

In `contracts/fund/src/events.rs:10`, the `event_withdraw` function emits the event type as "`withdraw_and_swap`". This is misleading because only a withdrawal action is performed without swapping.

Additionally, in `contracts/fund/src/handle.rs:358`, the comment is incorrect because it mentions that `mint_to_address` is being set, which is inaccurate because the code dispatches a `Bank` message instead.

Recommendation

We recommend updating the event type to "`withdraw`" and removing the incorrect amount.

Status: Resolved