Audit Report

Alpha

v1.0

July 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 Spectrum Protocol to perform a security audit of Alpha.

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/spectrumprotocol/alpha-contracts

Commit hash: `28857d208854516d1512a63f8476a93e5761a388`

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

Alpha provides a leveraged yield-farming product powered by smart contracts on the Terra blockchain. It offers a leveraged farm that allows leveraged liquidity provision with different asset weights and borrowed amounts on Astroport, a borrowed farm that allows one-sided liquidity provision, as well as Homora Bank which connects to Mars protocol's Red Bank and a lending bank and oracle contracts. LP tokens are automatically staked on Spectrum protocol.

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	Users will not get native assets back when borrowing them from borrowed farm	Major	Resolved
2	Attackers may delay price update transactions by feeders to manipulate oracle prices	Major	Resolved
3	New borrowers using Homora Bank will pay fees for older borrowers	Major	Acknowledged
4	Tokens with low price may have wrong price quoted by oracle due to loss of precision and price impact of probe amount	Major	Resolved
5	Wrong interest rate is used when withdrawing from lending bank, leading to small losses for other users	Major	Acknowledged
6	Adjusting positions up will fail if borrow asset is native but provide asset is CW20 or vice versa	Major	Resolved
7	Oracle uses median of recent (spot) prices, may be subject to spot price manipulation	Minor	Resolved
8	Swap amount calculation does not account for taxes	Minor	Acknowledged
9	Repayment may fail if amounts are small	Minor	Resolved
10	Swap calculation in leveraged farm uses hardcoded fee and will be wrong if the underlying fee is changed	Minor	Resolved
11	Lack of price validation in oracle might cause issues	Minor	Resolved
12	Config parameter validation missing in Homora Bank's init asset config	Minor	Resolved
13	Treating Luna as a special case for tax calculation may lead to problems with Terra protocol updates	Minor	Acknowledged
14	Reliance on assumed received amounts might cause issues in the future	Minor	Acknowledged
15	Failing swap simulation in leveraged farm may lead	Minor	Resolved

	to high arbitrage losses when providing liquidity		
16	Spot price queries from oracle may return wrong prices if neither asset is UST	Minor	Resolved
17	Interest rate calculation does not account for leap years/seconds	Minor	Resolved
18	Querying the lending bank's market list, user debt and user positiont may run out of gas	Minor	Acknowledged
19	Max diff value validation in borrowed farm is executed on previous value	Minor	Resolved
20	Leveraged farm incentivizes partial liquidation	Informational	Acknowledged
21	Allowances are increased without expiration	Informational	Resolved
22	Operators can add users with allowed assets that are not in asset config, which might cause issues in future versions	Informational	Resolved
23	Unbonded allowed assets `HashSet` in Homora Bank contract makes loading of users more gas-intensive with more entries	Informational	Acknowledged
24	Restricted config changes in Homora Bank contract through operator are silently ignored	Informational	Resolved
25	Staking `bond_msg` method does not support native assets despite signature	Informational	Resolved
26	Wrong error message for `AssetAlreadyInitialized` error in lending bank	Informational	Resolved
27	Unused code decreases maintainability	Informational	Resolved
28	Overflow checks not enabled for release profile	Informational	Resolved
29	Storing uncollateralized boolean in debt struct is inefficient	Informational	Resolved
30	Duplicate zero amount checks are inefficient	Informational	Resolved
31	Loading stored data twice is inefficient	Informational	Resolved
32	Updating the protocol admin address does not update existing IB tokens	Informational	Resolved
33	User health status borrowing variant contains fixed amount	Informational	Resolved

Code Quality Criteria

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

Detailed Findings

1. Users will not get native assets back when borrowing them from borrowed farm

Severity: Major

In the `callback_unbond_hook` function, stakers that borrow native assets should receive the swapped amount since the pair's `swap_msg` functions's `to` parameter is set to `Some(staker_addr.to_string())`, see `contracts/borrowed-farm/src/bond.rs:407`. For native assets though, there is a bug in the `swap_msg` functions that does not apply the `to` value in `packages/alpha-homora-v2/src/adapters/pair.rs:206`. Consequently, a swap of native assets would be called with `None` set as the `to` parameter, which would lead to the swapped funds being stuck in the borrowed farm contract.

Recommendation

We recommend replacing `to: None` with `to` in `packages/alpha-homora-v2/src/adapters/pair.rs:206`.

Status: Resolved

2. Attackers may delay price update transactions by feeders to manipulate oracle prices

Severity: Major

When feeders feed prices to the oracle contract, the `last_update_time` of the `PriceInfo` is set to the current block timestamp. When querying prices from the oracle, that `last_update_time` value is used in `contracts/oracle/src/contract.rs:265` to check whether the last updated price is too old. An attacker (or colluding validators) may exploit this by delaying transactions to shift price updates.

Suppose that a feeder sends a price update transaction in block 12345. An attacker may execute a DOS attack on the target node or the feeder's infrastructure to prevent the inclusion of the transaction until some point in the future (e.g. block 13000). In that later block 13000, the price will be accepted and will be considered up to date. That allows an attacker to shift prices around to their benefit.

Recommendation

We recommend adding the `last_update_time` into the `FeedPrice` message, and verifying in the handler that it's not too far away from the current block timestamp.

Status: Resolved

3. New borrowers using Homora Bank will pay fees for older borrowers

Severity: Major

During borrows and repays in the Homora Bank contract, the current debt state is updated in the `update_debt_state` function. In that function, the interest accrued within the Red Bank is calculated, and a proportional Homera fee is stored in `state.homora_fee`. When a user now repays their loan, they will have to pay that `homora_fee` proportionally to the amount of debt they repay. The current mechanism implemented in `contracts/homora-bank/src/contract.rs:462-464` does not account for the duration a user has been borrowing though, which implies that new users pay the fees for previous users.

As an example, suppose there is a current debt of `100` in the Homora Bank and a `homora_fee` of `10`. Now a user is borrowing `100` tokens and immediately repays these `100` tokens in the same block. The current logic will charge that user a proportional fee of `10 * 100 / 200 = 5` tokens.

Recommendation

We recommend introducing global and user-based fee indexes which will be updated whenever users interact with the contract. That will allow proper accounting of fees such that new users only pay for their portion of the fee.

Status: Acknowledged

The Alpha team states that this behavior is intended.

4. Tokens with low price may have wrong price quoted by oracle due to loss of precision and price impact of probe amount

Severity: Major

When calculating the median price in the oracle in `contracts/oracle/src/contract.rs:297`, the price is multiplied by the `price_multiplier`, which effectively truncates any decimals beyond 6 decimal places. This is a major issue for tokens that have very low prices and many decimal places. For

example, a token with 18 decimal places and a current price of `0.0000018` would be wrongly represented within the Alpha protocol with a price of `0.000001`.

Additionally, the probe amount may lead to too much of a price impact for tokens with a low token price.

Recommendation

We recommend setting the `DEFAULT_TOKEN_DECIMALS` to `18`, which is a value commonly found in tokens. We also recommend specifying a distinct probe amount per asset.

Status: Resolved

5. Wrong interest rate is used when withdrawing from lending bank, leading to small losses for other users

Severity: Major

During the `execute_withdraw` function of the lending bank contract, interest rates are updated to reflect the new balance after withdrawal in `contracts/lending-bank/src/contract.rs:701`. With the linear interest model, a withdrawal implies that utilization goes up, which means that both the borrow rate and the liquidity increase.

Then, in line `707`, the balance after withdrawal is calculated. Within that calculation in the `get_updated_liquidity_index` function, the new liquidity rate is used to update the liquidity index. Since the liquidity rate went up, the liquidity index is higher than it should be. This implies that the user will have fewer IB tokens burned than they should have. The protocol and hence other users will take the difference.

The impact of this issue depends on the activity of the protocol – with high amounts locked, small burned amounts and frequent interactions that update interest rates, the impact of a wrong index is rather minor. If there are big withdrawals and less frequent interactions though, there can be material losses for other users.

Recommendation

We recommend updating interest rates with the previous utilization rate before withdrawal, i.e. calling `update_interest_rates` without the `withdraw_amount`.

Status: Acknowledged

6. Adjusting positions up will fail if borrow asset is native but provide asset is CW20 or vice versa

Severity: Major

In the `increase_position_msgs` method in `contracts/borrowed-farm/src/adapters/leveraged_farm.rs:73`, there is a wrong check whether `provide_asset_info` is a native asset, while `borrow_asset_info` is sent in line `75`.

This is no issue if both tokens are either native or both are CW20 tokens. If they are of different asset types though, no funds of `borrow_asset_info` will be sent or no allowance will be increased, which will lead to a failure of the `execute_adjust_position` function when positions are increased in `contracts/borrowed-farm/src/bond.rs:557`.

Recommendation

We recommend replacing `provide_asset_info` with `borrow_asset_info` in line `73`.

Status: Resolved

7. Oracle uses median of recent (spot) prices, may be subject to spot price manipulation

Severity: Minor

Different functions in the protocol query the oracle contract for asset prices. Each asset has different price sources, and the oracle returns the median of those prices in the `query_asset_price` function in `contracts/oracle/src/contract.rs:297`. All of the price sources use a single stored or queried price. Some of those price sources use spot prices. Spot prices can be manipulated, for example by swapping large amounts of tokens before using the Alpha contracts. Depending on the configuration of an asset, this may allow manipulation of the median price, and hence may be exploited by an attacker.

We classify this issue as minor since spot prices are only supposed to be used for testing.

Recommendation

We recommend enforcing a minimum number of price sources in order for a query to succeed. We also recommend adding lagged or time weighted average prices to make price manipulation of spot prices more expensive and discourage attacks.

Status: Resolved

8. Swap amount calculation does not account for taxes

Severity: Minor

The `get_swap_amount` function is called in `contracts/leveraged-farm/src/deposit.rs:81` and `102` on the amounts prior to deducting tax, which happens in lines `84` and `104`, respectively. That implies that the swap calculation will be slightly off, and the small difference will be arbitrated away by external actors. While the impact of this difference is small, it will take away from a user's yield and is easily prevented.

We consider this issue to be minor since the impact is relatively small.

Recommendation

We recommend deducting taxes on the amounts that are passed into the `get_swap_amount` function.

Status: Acknowledged

9. Repayment may fail if amounts are small

Severity: Minor

During repayment, in the `callback_repay` function, a swap happens between asset A and asset B to settle outstanding debt from bank A and bank B. Those swaps have a condition to swap a minimum of 1000 tokens in `contracts/leveraged-farm/src/withdraw.rs:181` and `197`. If that minimum is swapped, the remaining amount of the bid asset might not suffice though to repay the other loan.

Recommendation

We recommend returning an error instead to allow users to resolve those issues.

Status: Resolved

10. Swap calculation in leveraged farm uses hardcoded fee and will be wrong if the underlying fee is changed

Severity: Minor

The `get_swap_amount` function in `contracts/leveraged-farm/src/deposit.rs:43` assumes that the underlying pool swap fee is 0.3%. That value is hardcoded in the formula and hence calculated swap amounts will be wrong if the underlying swapping protocol changes their fee.

The same issue exists in the `execute_adjust_position` function in `contracts/borrowed-farm/src/bond.rs:527`.

Recommendation

We recommend adding a `fee` config value that is used in the formula to update the fee if needed.

Status: Resolved

11. Lack of price validation in oracle might cause issues

Severity: Minor

Prices from feeders are currently not validated. Feeders could (accidentally) set a price to zero in `contracts/oracle/src/contract.rs:133`. That causes inconsistencies in the protocol, for example in health checks of a position.

Recommendation

We recommend validating that a price is not set to zero.

Status: Resolved

12. Config parameter validation missing in Homora Bank's init asset config

Severity: Minor

In the `execute_init_asset_config` function of the Homora Bank, configuration parameters are not validated in `contracts/homora-bank/src/contract.rs:175-180`. If those parameters are set to invalid values, the protocol might not work as expected – for example if the `collateral_factor` is set to zero.

Additionally, neither the `execute_init_asset_config` nor the `execute_update_asset_config` function currently validate the `fee_rate` or the `liquidation_bonus`.

We consider this to be a minor issue since it can only be caused by the owner and is recoverable.

Recommendation

We recommend adding validation to the `execute_init_asset_config` function, as is done in the `execute_update_asset_config` function. We also recommend adding validation the the `fee_rate` and `liquidation_bonus` parameters.

Status: Resolved

13. Treating Luna as a special case for tax calculation may lead to problems with Terra protocol updates

Severity: Minor

In `packages/alpha-homora-v2/src/adapters/asset.rs:318`, Luna is treated as a special case for tax calculations, with a hard-coded zero tax value. However, this might lead to inconsistencies if Terra changes the Luna tax policy in a future protocol update. In such a case, the contract would pay the tax, misusing funds.

Recommendation

We recommend treating Luna the same as other native tokens and querying the tax rate from Terra.

Status: Acknowledged

14. Reliance on assumed received amounts might cause issues in the future

Severity: Minor

In several places in the codebase, the implicit assumption is made that another contract is sending a requested amount. Even if correct today, that assumption might be wrong in a future update of a dependent contract, for example, if fees are introduced that reduce the sent amount.

An example of this can be found in the Homora Bank contract, where an asset is borrowed from the bank in `contracts/homora-bank/src/contract.rs:387`. Then the asset is sent to the borrower in line `389`, without verifying that the requested amount has actually been received.

Recommendation

We recommend calculating the actually received amount by querying the contract's balance before and in a reply after sending a message.

Status: Acknowledged

15. Failing swap simulation in leveraged farm may lead to high arbitrage losses when providing liquidity

Severity: Minor

If the swap simulation in `contracts/leveraged-farm/src/deposit.rs:89` or `109` fails, a zero value is used. Swapping is then skipped in lines `91` or `111`, but the liquidity is still provided and the position increased. This may cause liquidity provision with a ratio that is far away from actual pools of the AMM, and the difference will likely be arbitraged away by external parties, leading to a loss of value.

Recommendation

We recommend returning an error and reverting an increase of a position if the swap simulation fails.

Status: Resolved

16. Spot price queries from oracle may return wrong prices if neither asset is UST

Severity: Minor

In the `query_spot_price` function, the implicit assumption is made in `contracts/oracle/src/contract.rs:331-335` that one of the two assets of the pair is UST. If that's not the case, the price will always be quoted in `assets[1]`. This is problematic, since a misconfiguration of the pair would lead to the protocol executing with wrong prices, rather than returning errors.

Recommendation

We recommend checking whether `assets[1]` is UST in line `333`, and returning an error if it is not.

Status: Resolved

17. Interest rate calculation does not account for leap years/seconds

Severity: Minor

The `SECONDS_PER_YEAR` constant in `contracts/lending-bank/src/interest_rates.rs:20` has a value of `31536000` seconds, which corresponds to 365 days (a non-leap year). This implies that in leap years and when leap seconds are used, the interest rate will be slightly higher than expected.

Recommendation

We recommend averaging out leap years and seconds by using a value of `365.256363004` days which corresponds to `31558150` seconds per year.

Status: Resolved

18. Querying the lending bank's market list, user debt and user positiont may run out of gas

Severity: Minor

The `query_markets_list`, `query_user_debt`, and `get_user_asset_positions` functions contain unbounded iterations over all stored markets in `contracts/lending-bank/src/contract.rs:1079`, `1104`, and `contracts/lending-bank/src/accounts.rs:73`. That may run out of gas.

Recommendation

We recommend adding pagination to the queries or enforcing an upper limit on the number of markets that can added.

Status: Acknowledged

The Alpha team states that they intend to only use a low number of markets.

19. Max diff value validation in borrowed farm is executed on previous value

Severity: Minor

In the `execute_update_position_parameters` function in `contracts/borrowed-farm/src/contract.rs:297`, `validate_percentage` is called with the previous `max_diff` value, not the newly set one. That allows a `max_diff` value that is bigger than 1.

Recommendation

We recommend swapping lines `297` and `298`.

Status: Resolved

20. Leveraged farm incentivizes partial liquidation

Severity: Informational

During liquidation, a liquidation bonus is applied for each asset and the LP tokens in `contracts/leveraged-farm/src/health.rs:256-263`. Due to the condition in line `267`, a full liquidation would imply that no bonus is paid. Rational liquidators would try to maximize the bonus by only partially liquidating the position.

Recommendation

We recommend clearly documenting the fact that leveraged farm is incentivizing partial liquidations.

Status: Acknowledged

21. Allowances are increased without expiration

Severity: Informational

In `contracts/borrowed-farm/src/adapters/leveraged_farm.rs:66` and `84`, allowances are increased without an expiration. Even though the allowance is only increased by the amount that will be withdrawn, It is best practice to expire allowances in the same block whenever interacting with another smart contract.

Recommendation

We recommend adding an expiration of the current block height by setting `expires: Some(Expiration::AtHeight(env.block.height + 1))`.

Status: Resolved

22. Operators can add users with allowed assets that are not in asset config, which might cause issues in future versions

Severity: Informational

In the `execute_register_user` function, the operator can register arbitrary assets for a user in `contracts/homora-bank/src/contract.rs:261`, with no check against `ASSET_CONFIGS` whether the asset is already initialized.

While allowed assets for a user that are not in `ASSET_CONFIGS` cannot be borrowed, this still leaves room for misconfiguration and may become an issue in a future version of the smart contract.

The same issue exists in the `execute_update_user` function.

Recommendation

We recommend adding a check to verify that all allowed assets are in `ASSET_CONFIGS`.

Status: Resolved

23. Unbonded allowed assets `HashSet` in Homora Bank contract makes loading of users more gas-intensive with more entries

Severity: Informational

The Homora Bank contract contains a `HashSet` `allow_assets` for every user. Whenever a user borrows an asset, it is checked in `contracts/homora-bank/src/contract.rs:355` whether the asset exists in `allow_assets`. While lookups within a `HashSet` are efficient, every time the user is loaded, the whole `HashSet` will be deserialized, which means that gas consumption increases with more assets. Eventually, loading a user with many entries could even run out of gas.

We classify this issue as only informational since only the operator can register users with an asset list or update the allowed assets.

Recommendation

We recommend limiting the number of allowed assets. Alternatively, the `allow_assets` `HashSet` could be removed from the `User` struct, and instead, a separate storage map with the user's address and the asset reference as a key could be used.

Status: Acknowledged

The Alpha team states that this is not a concern since the number of allowed assets will regularly be 2 and maximally 4.

24. Restricted config changes in Homora Bank contract through operator are silently ignored

Severity: Informational

The Homora Bank contract's `execute_update_asset_config` function allows both the owner and the operator to update config values. The operator has fewer permissions than the owner, and can only update `borrow_enabled` and `repay_enabled`. If the operator is providing other values than these two in the `UpdateAssetConfig` struct, those other values are silently ignored. That might be confusing to the operator. It is best practice to return an error if insufficient permissions exist, rather than ignoring provided values.

Recommendation

We recommend returning an error if an operator tries to update values they have no permission to update.

Status: Resolved

25. Staking `bond_msg` method does not support native assets despite signature

Severity: Informational

The `bond_msg` method in `packages/alpha-homora-v2/src/adapters/staking.rs:49` accepts native assets through the asset parameter of type `Asset`, but the implementation only supports CW20 assets.

While this is not a problem for the current usage of the method for LP token staking, which are all CW20s, future changes or external parties using this code might run into this limitation.

Recommendation

We recommend either only accepting CW20 assets as arguments, or implementing support for staking of native assets. At the minimum, we suggest returning an error if a native asset is provided.

Status: Resolved

26. Wrong error message for `AssetAlreadyInitialized` error in lending bank

Severity: Informational

The `AssetAlreadyInitialized` error in `contracts/lending-bank/src/error.rs:41` contains the wrong error message `"User's health factor can't be less than 1 after withdraw"`.

Recommendation

We recommend updating the error message.

Status: Resolved

27. Unused code decreases maintainability

Severity: Informational

In several places in the codebase, unused code exists. One example is the `ContractError` enum in `contracts/oracle/src/error.rs`, which contains variants that are not utilized. Unused code makes the codebase more complex and hence negatively affects maintainability.

Recommendation

We recommend removing unused code.

Status: Resolved

28. Overflow checks not enabled for release profile

Severity: Informational

The following packages and contracts do not enable `overflow-checks` for the release profile:

- `contracts/ib-token/Cargo.toml`
- `contracts/lending-bank/Cargo.toml`
- `contracts/oracle/Cargo.toml`
- `packages/alpha-homora-v2/Cargo.toml`
- `packages/astroport/Cargo.toml`
- `packages/chainlink-aggregator/Cargo.toml`
- `packages/spectrum_protocol/Cargo.toml`

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

29. Storing uncollateralized boolean in debt struct is inefficient

Severity: Informational

The `uncollateralized` boolean value of the `Debt` struct for a user and asset is currently set in the `execute_update_uncollateralized_loan_limit` function in `contracts/lending-bank/src/contract.rs:557` and in the `execute_borrow` function in lines `801` and `842`. It is always `true` if a positive `UNCOLLATERALIZED_LOAN_LIMITS` entry exists for the user and asset, and always `false` otherwise. Consequently, it's unnecessary to store the value.

Recommendation

We recommend removing the `uncollateralized` boolean value from the `Debt` struct and checking whether the `UNCOLLATERALIZED_LOAN_LIMITS` entry is positive instead.

Status: Resolved

30. Duplicate zero amount checks are inefficient

Severity: Informational

Amounts are checked to not be zero multiple times in the same functions:

a) In `contracts/ib-token/src/contract.rs:238` and in the transfer in `245`.
b) In `contracts/ib-token/src/contract.rs:146` and in the transfer in `149`.

Recommendation

We recommend removing duplicate checks.

Status: Resolved

31. Loading stored data twice is inefficient

Severity: Informational

In `contracts/lending-bank/src/contract.rs:838-843`, debt is loaded which already happened in lines `797-802`. This is inefficient.

Recommendation

We recommend loading debt only once.

Status: Resolved

32. Updating the protocol admin address does not update existing IB tokens

Severity: Informational

In `contracts/lending-bank/src/contract.rs:253-354`, it is possible to update the `protocol_admin_address`. That does not update the admin of any already instantiated IB tokens though, which can lead to inconsistencies.

Recommendation

We recommend clearly documenting the need to manually update the admin of any existing IB token.

Status: Resolved

33. User health status borrowing variant contains fixed amount

Severity: Informational

In `contracts/lending-bank/src/accounts.rs:50`, the `UserHealthStatus::Borrowing` enum variant contains a value that is hardcoded to always be `Decimal::one()`.

Recommendation

We recommend removing the value.

Status: Resolved