Audit Report

Kinetic

v1.0

April 7, 2022

Table of Contents

Table of Contents **2**

License **4**

Disclaimer **4**

Introduction **6**

Purpose of This Report 6

Codebase Submitted for the Audit 6

Methodology 7

Functionality Overview 7

How to Read This Report **8**

Summary of Findings **9**

Code Quality Criteria 10

Detailed Findings **11**

Late depositors would get the same amount of dividend as early depositors, causing an unfair dividend allocation 11

Malicious users can trick the system into receiving more yield than entitled 11

Users are unable to withdraw all funds from vault contract 12

Updating user's collateral debt position causes loss of yield 12

Users are unable to withdraw funds once phaser is started 13

More synthetic tokens are burned than intended 13

Claim message in phaser contract withdraws more UST than the user should own, leading to a loss of funds for other users 14

Anchor exchange rate used might be out of date due to missing block height argument 14

Incorrect variable passed as address will cause harvest operation to fail 15

Updating collateralization limit config causes undercollateralized loans 15

Updating synthetic token address would cause state inconsistency 16

Vault contract's total debt is not decreased when repaying debt 16

Duplicate account creation would inflate total supply 17

Misconfigured decimal values might cause underflow issues 17

Incorrect phase period would cause division by zero panic 17

Burn execution would cause a corruption of TOTAL_SUPPLY state 18

Incorrect comments found in codebase 18

Incorrect total_deposited event emitted 19

Outstanding TODOs are present in the codebase 19

Overflow checks not set for release profile 19

When querying exchange_rate from adapter, the input_denom parameter is not used 20

Usage of unwrap can lead to panic execution 20
unwrap_or should only be used with compile time evaluated parameters 21
Misleading event emitted when setting unlimited minting capacity 21
CW2 version information uses wrong contract name 21
Misleading error message might confuse users 22

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 Kinetic Labs LTD to perform a security audit of the Kinetic 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/kinetic-money/core

Commit hash: `b52b2b3223cacc4fb40eecaf4ff5cec6c81f811d`

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

Kinetic money is a protocol on the Terra blockchain that allows users to deposit funds in return for a loan that repays itself. In order to fulfill its purpose, the protocol takes advantage of the yield generated by deposits to reduce the debt position.

The audited smart contracts implement the vault, the phaser, the anchor adapter, the coordinator, and the behaviors of the protocol's tokens.

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	Late depositors would get the same amount of dividend as early depositors, causing an unfair dividend allocation	Critical	Resolved
2	Malicious users can trick the system into receiving more yield than entitled	Critical	Resolved
3	Users are unable to withdraw all funds from vault contract	Critical	Resolved
4	Updating user's collateral debt position causes loss of yield	Critical	Resolved
5	Users are unable to withdraw funds once phaser is started	Critical	Resolved
6	More synthetic tokens are burned than intended	Critical	Resolved
7	Claim message in phaser contract withdraws more UST than the user should own, leading to a loss of funds for other users	Critical	Resolved
8	Anchor exchange rate used might be out of date due to missing block height argument	Major	Resolved
9	Incorrect variable passed as address will cause harvest operation to fail	Major	Resolved
10	Updating collateralization limit config causes undercollateralized loans	Major	Resolved
11	Updating synthetic token address would cause state inconsistency	Major	Resolved
12	Vault contract's total debt is not decreased when repaying debt	Minor	Resolved
13	Duplicate account creation would inflate total supply	Minor	Resolved
14	Misconfigured decimal values might cause underflow issues	Minor	Resolved
15	Incorrect phase period would cause division by zero panic	Minor	Resolved

16	Burn execution would cause a corruption of `TOTAL_SUPPLY` state	Minor	Resolved
17	Incorrect comments found in codebase	Informational	Resolved
18	Incorrect `total_deposited` event emitted	Informational	Resolved
19	Outstanding TODOs are present in the codebase	Informational	Resolved
20	Overflow checks not set for release profile	Informational	Resolved
21	When querying `exchange_rate` from adapter, the `input_denom` parameter is not used	Informational	Resolved
22	Usage of `unwrap` can lead to panic execution	Informational	Resolved
23	`unwrap_or` should only be used with compile time evaluated parameters	Informational	Acknowledged
24	Misleading event emitted when setting unlimited minting capacity	Informational	Resolved
25	CW2 version information uses wrong contract name	Informational	Resolved
26	Misleading error message might confuse users	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium-High	-
Level of documentation	Medium-High	-
Test coverage	Medium-High	cargo-tarpaulin reports a test coverage of 86%.

Detailed Findings

1. Late depositors would get the same amount of dividend as early depositors, causing an unfair dividend allocation

Severity: Critical

In `contracts/core/phaser/src/contract.rs:211`, new users that stake their `kUST` tokens will have their `last_dividend_points` set to 0. This is problematic since the value of `last_dividend_points` is used to determine the amount of dividends a user is entitled to claim as seen in `contracts/core/phaser/src/util.rs:72-78`. This would cause late depositors to claim the same dividend allocation as early depositors, resulting in unfair dividend allocation and possibly insufficient funds in the contract.

This issue is also present in the `vault` contract in `contracts/core/vault/src/contract.rs:262`. Here, the value of `last_accumulated_yield_weight` is used to calculate the yield users are entitled to earn in `contracts/core/vault/src/cdp.rs:40-53`. As above, late depositors would be able to claim more yield than intended.

Recommendation

We recommend setting the value of `last_dividend_points` to `cfg.total_div` in `contracts/core/phaser/src/contract.rs:211`. As for the `vault` contract, we recommend setting the value of `last_accumulated_yield_weight` to `cfg.accumulated_yield` in `contracts/core/vault/src/contract.rs:262`.

Status: Resolved

2. Malicious users can trick the system into receiving more yield than entitled

Severity: Critical

In `contracts/core/vault/src/contract.rs:252-256`, existing depositors will have their `cdp.total_deposited` value increased based on the amount they deposited without allocating the yield beforehand. This is problematic because the yield earned is based on the user's deposited amount as seen in `contracts/core/vault/src/cdp.rs:50`. Consequently, a malicious user can deposit funds repeatedly into the vault contract and withdraw them after some time to claim more yield than they should have received.

Recommendation

We recommend processing yield allocation before increasing the user's deposited amount, i.e. calling `cdp_update` before `contracts/core/vault/src/contract.rs:253`.

Status: Resolved

3. Users are unable to withdraw all funds from vault contract

Severity: Critical

Users are able to call the `Redeem` message in order to withdraw `UST` or `aUST` from the vault contract. Before that, the user's collateral debt position (CDP) is checked to verify the user's position is not undercollateralized. However, users will be unable to withdraw the max available amount of funds since the `is_cdp_healthy_after_withdraw` function verifies that the user's CDP is lower than the collateralization limit by using `Decimal256::from_ratio` as seen in `contracts/core/vault/src/cdp.rs:21`. If the user attempts to withdraw all available funds, the `value` passed would become 0 and eventually cause a division by 0 panic.

Recommendation

We recommend allowing the user to withdraw all their funds without causing a division by 0 panic.

Status: Resolved

4. Updating user's collateral debt position causes loss of yield

Severity: Critical

In `contracts/core/vault/src/cdp.rs:29`, user's `cdp.total_credit` value is updated when the earned yield is higher than user's debt. If the user has an existing `cdp.total_credit` balance, it would be overwritten due to the latest value being directly set instead of increased along with the existing balance. This would cause a portion of the user's earned yield to be stuck in the contract.

Recommendation

We recommend increasing the user's existing `cdp.total_credit` balance via modifying `contracts/core/vault/src/cdp.rs:29` to:

```
cdp.total_credit += earned_yield.sub(cdp.total_debt);
```

Status: Resolved

5. Users are unable to withdraw funds once phaser is started

Severity: Critical

In `contracts/core/phaser/src/util.rs:63`, the `update_account` function attempts to deduct users' entitled dividend from `cfg.unclaimed_div` value. This value can only be increased when there's an increase in `UST` allocations as seen in line `54`. However, the increased amount comes from `cfg.total_div` which means the amount will never be enough to deduct user's entitled dividend, which will cause an underflow error. To illustrate the scenario:

1. Alice stakes 100 `kUST` via `ReceiveMsg::Stake`, this would cause `user.deposited_synth_tokens` and `cfg.total_deposited_synth_tokens` to be 100.
2. 200 `UST` is sent via `Distribute` message which causes `cfg.total_deposited_base_tokens` and `cfg.buffer` to be 200.
3. Bob decides to stake 50 `kUST` which would cause `run_phaser` to execute. Assuming `cfg.phase_period` has passed, the contract will enter line `53-54` with `cfg.total_div` and `cfg.unclaimed_div` value as 2 (200/100 = 2).
4. At this point, there's a high possibility that no one can withdraw their funds since the phaser is executed.
5. Alice decides to withdraw her tokens by calling `ExecuteMsg::Unstake` which will call `update_account`. In result, the `owing` amount returned from `dividends_owing` function would be 200 (100*2 = 200, see lines `73-75`). Since the value of `cfg.unclaimed_div` is 2 while the `owing` value is 200, this would cause an underflow error in line `63`.

This issue also affects other entry points that call `update_account` function, which are `Phase`, `Stake`, and `Unstake`.

Recommendation

We recommend modifying the implementation to process user dividends correctly.

Status: Resolved

6. More synthetic tokens are burned than intended

Severity: Critical

In `contracts/core/phaser/src/contract.rs:376`, the amount of synthetic tokens burned depends on the balance of `user.realized_tokens` and not on the currently phased amount. This would cause more synthetic tokens being burned than the intended amount, which opens the possibility that there are not enough synthetic tokens in the phaser contract. As a result, this would cause a loss of funds for the protocol.

Recommendation

We recommend supplying the correct amount when burning synthetic tokens in `contracts/core/phaser/src/contract.rs:376` by replacing the amount with `pending.into()`.

Status: Resolved

7. Claim message in phaser contract withdraws more UST than the user should own, leading to a loss of funds for other users

Severity: Critical

In `contracts/core/phaser/src/contract.rs:309-313`, when a user decides to claim their phased UST, the contract withdraws the amount in aUST without calculating the exchange rate. Due to the fact that aUST usually trades above UST, this may lead to a withdrawal of a greater amount of UST from the adapter than the user should be able to claim. This causes a loss of funds for the whole protocol.

Recommendation

We recommend querying the UST/aUST exchange rate for the current block and create the withdrawal message with `user.realized_tokens.div(exchange_rate)` instead of `user.realized_tokens`.

Status: Resolved

8. Anchor exchange rate used might be out of date due to missing block height argument

Severity: Major

The Anchor exchange rate is queried from the Anchor contracts, through the utility function `epoch_state`, defined in `contracts/core/adapters/anchor/src/msg.rs:102-114`. However, no block height argument is supplied. Without the block height argument, Anchor returns a raw exchange rate from stored values without accruing interest since the last update, which means an outdated exchange rate is returned. Using this value could lead to slightly incorrect calculations.

Recommendation

We recommend passing the current block height (`env.block.height`) in `contracts/core/adapters/anchor/src/msg.rs:108` to return the correct exchange rate,

Status: Resolved

9. Incorrect variable passed as address will cause harvest operation to fail

Severity: Major

In `contracts/core/vault/src/utils.rs:37`, when calling the `balance_of` function in order to query for the `yield_token_balance`, `config.yield_token` is passed as the second parameter of the function.

Since `config.yield_token` doesn't contain the token contract address but it represents the token denom which is `aUST`, it will cause the `execute_harvest` function to fail.

Recommendation

We recommend passing `config.yield_address` when calling the `balance_of` function in `contracts/core/vault/src/utils.rs:37`.

Status: Resolved

10. Updating collateralization limit config causes undercollateralized loans

Severity: Major

In `contracts/core/vault/src/contract.rs:604`, the validation attempts to verify the new `collateralization_limit` value to be 2 or 3. This would mean the collateralization limit can only be updated from 200% to 300%, essentially allowing user's CDP to be under-collateralized.

As an example of collateralization limit being 200%, a user that deposits 1000 UST as collateral would be able to withdraw 2000 kUST worth of debt. While the initial collateral will continue generating yield to repay the debt, it would take approximately 10 years for it to be fully repaid (`2000/(1000*19.5%) = 10.256..`). In addition to the long repayment period, kUST might encounter a high selling pressure since users can freely withdraw a loan twice the amount of their collateral without the commitment of repaying them. A user might go even further by looping the whole operation and siphoning all the available UST in the phaser contract, possibly causing insufficient funds for other users to redeem.

Recommendation

We recommend reworking the `validate_basic` method of the `Config` struct in order to validate the `collateralization_limit` value during contract instantiation and config updates. Other than that, consider modifying the constants `MAX_COLLATERALIZATION_LIMIT` and `MIN_COLLATERALIZATION_LIMIT` to 1 and 0 respectively.

Status: Resolved

11. Updating synthetic token address would cause state inconsistency

Severity: Major

In `contracts/core/phaser/src/contract.rs:498`, the address of the synthetic token can be updated to a different value. If the synthetic token address is updated while the contract has an existing `cfg.total_deposited_synth_tokens` value, it would cause an inconsistency between the contract's state and the actual token balance held in the contract. As a result, this would cause a series of problems such as users being unable to withdraw their tokens due to insufficient contract balance.

Recommendation

We recommend removing the functionality to update the synthetic token address.

Status: Resolved

12. Vault contract's total debt is not decreased when repaying debt

Severity: Minor

When repaying debt in the vault contract, only the user's debt value in their collateral debt position is reduced in `contracts/core/vault/src/util.rs:110`, but not the `cfg.total_debt`. This would cause the config query message to return an incorrect total debt amount as seen in `contracts/core/vault/src/contract.rs:716`.

Recommendation

We recommend deducting the contract's `total_debt` value in `repay_cw20` and `repay_native`.

Status: Resolved

13. Duplicate account creation would inflate total supply

Severity: Minor

In `contracts/token/cw20-token/src/contract.rs:74-78`, duplicate accounts are not verified when creating initial accounts during the contract instantiation phase. If the same account address is passed twice, the account's balance would be overwritten via `BALANCES.save`, but `total_supply` would still record the balance amount of both. As a result, the token's total supply would be inflated.

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

Recommendation

We recommend returning an error if duplicate accounts exist in the `msg.initial_balances` vector that's passed into `create_accounts`.

Status: Resolved

14. Misconfigured decimal values might cause underflow issues

Severity: Minor

In `contracts/token/coordinator/src/contract.rs`, `reward_factor`, `dev_factor`, `charity_factor`, and `protocol_fee` are not validated to be equal or lower than `1.0`. If the values are misconfigured to be greater than `1.0`, they will cause underflow issues due to insufficient amount during deduction. For example, if the `charity_factor` is misconfigured to be above `1.0`, it would cause the `sweep` operation to fail in lines `174-176`.

Recommendation

We recommend verifying decimal values to be equal to or lower than `1.0` in lines `42`, `45`, `47`, `58`, `105-113`, and `143`.

Status: Resolved

15. Incorrect phase period would cause division by zero panic

Severity: Minor

In `contracts/core/phaser/src/util.rs:27`, `cfg.phase_period` is used as the divisor in the calculation of the buffer distribution amount. If its value is 0, it would cause a division by 0 panic, causing the `run_phaser` execution to fail.

Recommendation

We recommend verifying the value of `cfg.phase_period` to be non-zero in `contracts/core/phaser/src/contract.rs:56` and `513`.

Status: Resolved

16. Burn execution would cause a corruption of `TOTAL_SUPPLY` state

Severity: Minor

During the `execute_burn` function In `contracts/token/cw20-staking/src/contract.rs:89`, the `TOTAL_SUPPLY` state is not decreased when the user decides to burn tokens. This will cause the `Investment` query message to return an incorrect state value as seen in lines `291-297`.

This issue is also present in line `116` when a user attempts to execute the `BurnFrom` message. In contrast, during the `unbond` function in lines `214-238`, `TOTAL_SUPPLY` is correctly decreased.

Recommendation

We recommend only allowing users to burn their tokens via the `Unbond` message and deduct `TOTAL_SUPPLY` accordingly when a user attempts to burn tokens via `BurnFrom` message.

Status: Resolved

17. Incorrect comments found in codebase

Severity: Informational

During the audit engagement, several incorrect comments were found in the following lines which would negatively affect the code readability and maintainability:

- `contracts/core/vault/src/contract.rs:399`
- `contracts/token/community/src/contract.rs:70`

The first comment should indicate transferring `aUST` to the user while the second comment should be `KNTC` instead of `ANC`.

Recommendation

We recommend modifying the comments as stated above.

Status: Resolved

18. Incorrect `total_deposited` event emitted

Severity: Informational

In `contracts/core/phaser/src/contract.rs:171`, the `total_deposited` attribute attempts to emit the total deposited UST amount plus the received UST amount. However, the received UST amount is already included inside `cfg.total_deposited_base_tokens` as seen in line `161`.

Recommendation

We recommend emitting the correct amount in the event by replacing line `171` with `(cfg.total_deposited_base_tokens).to_string()`.

Status: Resolved

19. Outstanding TODOs are present in the codebase

Severity: Informational

During the audit engagement, several `TODO` comments were found in the following lines:

- `contracts/core/phaser/src/util.rs:248`
- `contracts/core/vault/src/util.rs:62`
- `contracts/core/vault/src/state.rs:12`
- `contracts/core/vault/src/state.rs:21`

This implies that the contract might still be under development and not yet ready for mainnet deployment.

Recommendation

We recommend resolving the `TODO` issues and/or removing them from the codebase.

Status: Resolved

20. Overflow checks not set for release profile

Severity: Informational

The following `Cargo.toml` files do not enable `overflow-checks` for the release profile:

- `contracts/token/cw20-staking/Cargo.toml`
- `contracts/token/cw20-token/Cargo.toml`
- `contracts/token/vesting/Cargo.toml`

Recommendation

Even though this check is implicitly applied to all packages from the workspace `Cargo.toml`, we recommend also explicitly enabling overflow checks in every individual package. That helps prevent unintended consequences when the codebase is refactored in the future.

Status: Resolved

21. When querying `exchange_rate` from adapter, the `input_denom` parameter is not used

Severity: Informational

In `contracts/core/adapters/anchor/src/contract.rs:56`, the `input_denom` parameter is not used when querying `ExchangeRate`.

Recommendation

We recommend removing that parameter from `QueryMsg::ExchangeRate` since the contract already uses the default `input_denom` from the `Config` struct.

Status: Resolved

22. Usage of `unwrap` can lead to panic execution

Severity: Informational

In `contracts/core/phaser/src/utils.rs:21`, the result of a `checked_sub` that is wrapped inside a `Result` is passed to `unwrap)`. The usage of this function is generally discouraged because it can lead the execution to panic without a developer friendly error message. Unwraps also cause the wasm execution to abort, which does not allow handling of the error from calling functions.

Recommendation

We recommend handling the error more gracefully matching the `Result` instead of letting the code panic.

Status: Resolved

23. `unwrap_or` should only be used with compile time evaluated parameters

Severity: Informational

In `contracts/token/coordinator/src/utils.rs:19`, the `checked_sub Result` is handled with `unwrap_or`. This type of unwrapping is discouraged when passing as a parameter a function that needs to be evaluated.

Recommendation

We recommend to use `unwrap_or_else()` when passing a parameter that contains a function that should be evaluated.

Status: Acknowledged

The Kinetic Money team prefers to leave `unwrap_or` and return 0.

24. Misleading event emitted when setting unlimited minting capacity

Severity: Informational

In `contracts/token/cw20-token/src/contract.rs:163`, the `cap` value is set to 0 when the minter has unlimited minting capacity. This will cause the event in line `167` to emit a `new_cap` of 0, which might confuse users since it indicates that the minter cannot mint any tokens, which is incorrect.

Recommendation

We recommend changing it to a more appropriate value, e.g. `unlimited`.

Status: Resolved

25. CW2 version information uses wrong contract name

Severity: Informational

The contract name for the CW2 version information in `contracts/token/cw20-token/src/contract.rs:26` states that the contract is called `crates.io:cw20-base`, which is the name of the CW20 base contract.

Recommendation

We recommend setting an appropriate name for the contract, e.g. `kinetic-cw20-token`.

Status: Resolved

26. Misleading error message might confuse users

Severity: Informational

In `contracts/core/vault/src/contract.rs:409`, the `InvalidDenom` error message would be returned if a user supplied a `denom` value that is not `cfg.base_token` or `cfg.yield_token`. The current error message returned is `Must send valid tokens to deposit` which might confuse users since they are performing a `Redeem` operation to withdraw funds from the contract.

Recommendation

We recommend changing the error message to be more generic, e.g. `Invalid denom provided`.

Status: Resolved