Audit Report

Margined Protocol Perpetuals

v1.0

October 28, 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 Staedter Limited to perform a security audit of Margined Protocol Perpetuals.

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/margined-protocol/perpetuals

Commit hash: `98b4389c0549b91e76824b0afa3043e33532a80d`

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 submitted code implements Margined Protocol Perpetuals, a decentralized perpetual protocol, and multichain margin engines for CosmWasm enabled blockchains. The audit scope includes the following contracts:

- Margined engine, responsible for managing user positions and collateral.
- Fee pool that accrues the fees generated by the protocol.
- Insurance fund to cover a shortfall in funding payments.
- Price feed to provide market data.
- A virtual automated market maker (VAMM) that enables users to take perpetual positions.

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 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.

Summary of Findings

No	Description	Severity	Status
1	Bad debt state is not recorded	Critical	Resolved
2	Incorrect accounting during liquidation	Major	Resolved
3	Bad debt reset upon realization	Major	Resolved
4	Possible inconsistencies when configuring decimal values	Major	Resolved
5	Margined engine cannot support native tokens as collateral during mainnet production	Major	Resolved
6	Emergency `ShutdownVamms` messages are not able to execute `SetOpen` transactions due to lack of permissions	Major	Resolved
7	Updating eligible collateral causes multiple consequences	Major	Resolved
8	Insurance fund beneficiary can be updated	Minor	Resolved
9	Incorrect comparison of negative values on the Integer library	Minor	Resolved
10	Suboptimal Access Controls implementation	Minor	Resolved
11	Arbitrarily long pause possible	Minor	Acknowledged
12	Margin deposit to non-existing positions will lead to loss of user funds	Minor	Resolved
13	Low decimals configured in virtual AMM might cause incorrect swap returns	Minor	Resolved
14	Configuring the maintenance margin ratio higher than the initial margin ratio might open up liquidation possibilities	Minor	Resolved
15	Beta version dependencies shouldn't be used in production	Minor	Resolved
16	Users can open positions with a fractional leverage	Minor	Resolved
17	Centralization risks	Minor	Acknowledged

18	Lack of configuration parameters validation	**Informational**	**Partially Resolved**
19	Lack of pauser role	**Informational**	**Resolved**
20	Sender does not need to be validated	**Informational**	**Resolved**
21	`validate_address` functionality can be removed	**Informational**	**Resolved**
22	`require_additional_margin` can be simplified for readability	**Informational**	**Resolved**
23	Confusing error messages and comments	**Informational**	**Resolved**
24	Multiple inefficient execution flows	**Informational**	**Resolved**
25	Outdated and unmaintained dependencies in use	**Informational**	**Resolved**
26	Additional funds sent to the contract are lost	**Informational**	**Resolved**
27	`CONTRACT_NAME` should include a namespace to be adherent to the CW2 specification	**Informational**	**Resolved**

Code Quality Criteria

Criteria	**Status**	**Comment**
Code complexity	**Medium**	-
Code readability and clarity	**Medium-High**	-
Level of documentation	**Low-Medium**	Some documentation sections are works in progress
Test coverage	**High**	cargo-tarpaulin reports a 98.11% code coverage

Detailed Findings

1. Bad debt state is not recorded

Severity: Critical

In `contracts/margined_engine/src/messages:185`, the `withdraw` function called when executing the `WithdrawMargin` message is updating the `bad_debt` attribute of the `state`.

As `state` is passed as a `&mut`, the `withdraw` function caller has in the `state` variable the updated `bad_debt` value.

The `state` is not stored though, hence the new value is available only in the current execution scope and cannot be retrieved in a subsequent execution. Consequently, the `bad_debt` information is never saved in the contract store, and the information is lost.

This implies that the `margined_engine` could be insolvent or with an insolvency risk without being aware. Additionally, the `State` query, which will be used to track the `margined_engine` status, would always report the `bad_debt` metric to be lower than the real accrued debt.

See Appendix-1 for a test case that shows this issue.

Recommendation

We recommend saving the `state` struct with the updated `bad_debt` attribute in the contract store.

Status: Resolved

2. Incorrect accounting during liquidation

Severity: Major

The margined engine contract incorrectly keeps track of the funds that should be requested or sent to the insurance funds contract upon liquidation.

In the case of `liquidation_fee > remain_margin.margin` in `contracts/margined_engine/src/reply.rs:458`, `remain_margin.margin` is not set to zero at the end of the if block, effectively being used first in the calculation of `bad_debt` and later on the transfer of remaining margin at line `474`. This results in the engine contract unexpectedly moving additional funds to the insurance fund contract.

In addition, if there is no previously recorded `state.bad_debt` and the engine contract doesn't have enough funds to pay a liquidator, the missing amount would be withdrawn twice from the insurance fund contract: The first during `realize_bad_debt` in line `468` and a second time during `withdraw` in line `479`.

Given the Bad debt state is not recorded issue, this could cause the Insurance Fund contract to be drained faster than expected, potentially causing insolvency issues if not watched carefully. Additionally, these extra transfers incremented gas costs on every transaction related to the affected feature.

See Appendix-3 for a test case that shows this issue.

Recommendation

The recommendation is divided into two parts:

- First, adding `remain_margin.margin = Uint128::zero();` to the end of the `if` block at `contracts/margined_engine/src/reply.rs:458`.
- Then, review the logic related to `bad_debt` realization and shortfall covering within `withdraw()` so the amount is only transferred once.

Status: Resolved

3. Bad debt is wrongly reset upon realization

Severity: Major

When the `realize_bad_debt` function in `contracts/margined_engine/src/utils.rs:81` is called with a non-zero `state.bad_debt` value, instead of updating it with any additional `bad_debt` and withdrawing funds from the insurance fund contract, it is reset to zero and no transfer from the insurance fund is requested. This implies losing track of any previous debt and not covering the new one given the lack of withdrawal from the insurance fund.

Consequently, the contract could be at an insolvency risk without the team being aware as the `bad_debt` value being reported would be lower than the real accrued debt.

Recommendation

The function should continue to accumulate the value of previously existing debt for tracking purposes and transfer funds to cover the added amount.

Status: Resolved

4. Possible inconsistencies when configuring decimal values

Severity: Major

In both margined engine and virtual AMM contracts, contract instantiators are required to specify decimal values as seen in `contracts/margined_engine/src/contract.rs:62-66` and `contracts/margined_vamm/src/contract.rs:40`. There is no validation that ensures both values are same and consistent with each other when configuring and updating the decimal values via `contracts/margined_engine/src/handle.rs:83`. If both contracts assume different decimal values, it will yield incorrect results when executing `SwapInput` and `SwapOutput`, potentially returning a higher or lower output amount than intended that directly affects trader's position size.

Moreover, the `MAX_ORACLE_SPREAD_RATIO` constant in `contracts/margined_vamm/src/contract.rs:26` is represented as `10%` with 8 decimal values, meaning that the intended decimal values should be 9 (`100_000_000` / `0.1` = `1_000_000_000`). Since the constant value cannot be modified, a misconfiguration of decimals not being 9 would cause `query_is_over_spread_limit` in `contracts/margined_vamm/src/query.rs:230` to return an incorrect spread limit exceeded result.

Besides that, native token decimals values are assumed as 6 as seen in `packages/margined_common/src/asset.rs:191`. This collides with the virtual AMM decimals requirement since it intends to accept 9 decimals to satisfy `MAX_ORACLE_SPREAD_RATIO` constant value.

Recommendation

We recommend applying the fixes below:

- Verifying both decimals values are the same when initializing margined engine and virtual AMM contract to make it consistent.
- Reworking the implementation to configure `MAX_ORACLE_SPREAD_RATIO` to dynamically set itself to `10%` of configured decimal to allow native tokens to be supported in the virtual AMM.
- Removing the margined engine functionality to update the decimal values as virtual AMM's decimal value cannot be updated.

Status: Resolved

5. Margined engine cannot support native tokens as collateral during mainnet production

Severity: Minor

In `packages/margined_common/src/validate.rs:28-33`, the `validate_eligible_collateral` functionality only recognizes `ujunox` and `uwasm` as supported native tokens when configuring and updating eligible collateral assets. Since `ujunox` prefix only exists in Juno Testnet and `uwasm` prefix originates from wasmd, the hardcoded native tokens will not be usable during production as they don't exist in the Juno mainnet.

Also, as not all native tokens follow the `utoken` convention and could have a custom decimal digits representation, the hardcoded `6u8` value in `packages/margined_common/src/asset.rs:191` could lead to possible wrong calculations.

Recommendation

We recommend modifying the implementation to allow the contract owner to configure supported native tokens during the instantiation of the contract.

Status: Resolved

6. Emergency `ShutdownVamms` messages are not able to execute `SetOpen` transactions due to lack of permissions

Severity: Major

In `contracts/margined_insurance_fund/src/handle.rs:78`, when handling `ShutdownVamms` messages, the execution is iterating through all the `VAMMs` registered in the `insurance_fund` in order to send `VammExecuteMessage::SetOpen { open: false }` messages to them. Those messages are wrapped in a `SubMsg` and have the purpose of pausing that specific `VAMM`.

However the `SetOpen` message handler has a guard at line `contracts/margined_vamm/src/handle.rs:98` that is only accepting transactions from its registered `Owner` that is not intended to be the `insurance_fund` contract. Consequently all the `SetOpen` `SubMsg` will fail due missing `Owner` role permissions.

Recommendation

We recommend allowing the `SetOpen` message to be executed by the `insurance_fund`.

Status: Resolved

7. Updating eligible collateral causes state inconsistencies

Severity: Minor

In `contracts/margined_engine/src/handle.rs:73-84`, the contract owner can update the eligible collateral, which also updates the decimal values accordingly. This is problematic for the following reasons:

Firstly, it would cause a state inconsistency issue. The actual balance held in the contract for the newly configured eligible collateral might be different than the recorded state. As a result, traders might be unable to withdraw excess margin due to insufficient balance.

Secondly, it causes a decimal collision issue as the virtual AMM that the margined engine contract interacts directly with is unable to have the decimal values updated. This causes inconsistencies between the two contracts and will be further elaborated on in the next section.

Lastly, it would cause incorrect ratios and liquidation fees if the newly configured decimal value is lower than the previous value. Since `initial_margin_ratio`, `maintenance_margin_ratio`, `partial_liquidation_margin_ratio` and `liquidation_fee` values use decimal value as limit (i.e. `0%` to `100%`), a lower decimal configured will cause ratios and liquidation fees to exceed the `100%` limit. For example, if the previous decimal value is 9 and the new eligible collateral uses 6 decimals, a configured `10%` ratio or liquidation fee would become `10000%`, which will likely cause all positions to be undercollateralized due to `contracts/margined_engine/src/handle.rs:284`.

We classify this issue as minor, since only the contract owner can cause it.

Recommendation

We recommend removing the functionality to modify eligible collateral assets and deploy multiple margined engine contracts if multiple collaterals need to be supported.

Status: Resolved

8. Insurance fund beneficiary can be updated

Severity: Minor

The insurance fund contract allowed its owner to update the beneficiary at any time as seen in `contracts/margined_insurance_fund/src/handle.rs:36`. A malicious insider or an attacker that compromised the keys of the legitimate administrator would be able to set themselves as the beneficiary and then withdraw the contract's funds.

We classify this issue as minor since only the owner can update the insurance fund beneficiary and it is assumed that the owner is a trusted party.

Recommendation

We recommend removing the possibility to update the insurance beneficiary to avoid unintended usage of the withdrawal functionality.

Status: Resolved

9. Incorrect comparison of negative values in the integer library

Severity: Minor

The signed integer implementation found at `packages/margined_common/src/integer.rs` includes comparison functions in lines `471` and `485`. In both cases, the result of comparing two negatives is not correctly evaluated.

As an example, if `-5` and `-2` are taken, `-5` would be evaluated to be bigger than `-2`. A test case showcasing the issue can be found in Appendix-5.

Although the potential impact of this issue could be high, no exploitable instance was actually found in the contract's within scope. Therefore this finding is classified as minor.

Recommendation

The implementation should be reviewed to evaluate the element with the smaller absolute value as bigger when the two items being compared are negative numbers.

Status: Resolved

10. Suboptimal Access Controls implementation

Severity: Minor

The contracts within scope implemented in-house access controls. Although no instance of broken controls or bypasses has been found, some details have not been considered.

The transfer of the owner role to a different account is implemented in a one-step fashion without confirmation from the receiving party. This could potentially cause a loss of access to the role in case a typo or mistake is made during the role transfer.

In addition, the Access Control logic that enforces these restrictions is duplicated across the handlers of the related functions, which negatively impacts the code's readability and maintainability, as it is error-prone.

Recommendation

We recommend making use of a well-known access controls implementation as `cw_controllers::Admin` (https://docs.rs/cw-controllers/0.14.0/cw_controllers/struct.Admin.html). This library actually implements remediation for the issues described above and undergoes deep testing from the community.

Status: Resolved

11. Arbitrarily long pause possible

Severity: Minor

The margined engine contract implements a pausing functionality as an emergency mechanism at `contracts/margined_engine/src/handle.rs:115`. However, the function didn't consider a maximum duration of this pausing, allowing for potentially infinite pauses that would lock the contract forever in case the owner would have lost access to their keys or become compromised.

Recommendation

We recommend implementing pausing mechanisms with time restrictions, either by pausing for a hard-coded duration or by letting the pauser specify a duration, which should be validated against a maximum.

Status: Acknowledged

The team states that if they add a timer on the pause duration, it would still be possible to resend pause messages at the end of the timer and continually pause the vAMM. Choosing a timer and a predetermined length is counterintuitive because any issues that require pausing might not be solved before the timer is up. If they allow consecutive pauses, that is the same as not having a timer.

Further on, the team states that if this is a security issue, then pausing can be controlled by governance - but this solution undermines the idea of pausing and unpausing the vAMM, because as stated in issue 20, it adds extra delays. Additionally, users engaging in governance may not understand when the vAMM should be unpaused.

An indefinite pause might seem bad, but it only prevents users from changing their positions. If other security issues are addressed, e.g. changing beneficiaries, then very little can be done during indefinite pauses. It would be against the interests of users (who trade on the protocol) and devs (who earn fees from trading on the protocol) to maintain an indefinite pause.

12. Margin deposit to non-existing positions will lead to loss of user funds

Severity: Minor

The margined engine contract allows users to deposit further margin on existing positions. The target position is retrieved at `contracts/margined_engine/src/handle.rs:376`, but If no position is matched - for example, if the user provided the wrong vAMM - it would return a default one and then add margin to it.

This would most likely not be the user's intention, as adding a margin to an empty position is not beneficial. As a result, the user's funds would get stuck in the contract because the `withdraw_margin` functionality verifies the provided virtual AMM address must be registered in line `403`.

Recommendation

We recommend validating the retrieved position in the `deposit_margin` function. If the retrieved position is the default one, an error should be thrown stating that there are no existing positions for that combination of trader and vAMM.

Additionally, we recommend verifying the virtual AMM is registered using `require_vamm(deps.as_ref(), &config.insurance_fund, &vamm)?;` to prevent user depositing into an invalid virtual AMM contract.

Status: Resolved

13. Low decimals configured in virtual AMM might cause incorrect swap returns

Severity: Minor

In `contracts/margined_vamm/src/handle.rs:328` and line `371`, the `base_asset_bought` and `quote_asset_sold` are incremented by one if the calculated remainder is not 0. This might cause incorrect swap output to be returned if the configured decimal value is low. A test case showcasing the issue can be found in Appendix-4.

We consider this a minor issue because only the contract owner can configure decimals.

Recommendation

We recommend applying a minimum limit of decimal values that can be configured in the contract.

Status: Resolved

14. Configuring the maintenance margin ratio higher than the initial margin ratio might open up liquidation possibilities

Severity: Minor

In `contracts/margined_engine/src/handle.rs:160` and `284`, the `initial_margin_ratio` is validated when opening the position to ensure the position is overcollateralized while `maintenance_margin_ratio` is validated when liquidating a position to ensure the position is undercollateralized. Since both margin ratios can be configured as different values, configuring the maintenance margin ratio higher than the initial margin ratio would cause the trader's position to be liquidatable if they only satisfy the initial margin requirements.

For example, the initial margin ratio is configured to 5%, while the maintenance margin ratio is configured to 10%. If the trader opens a position with a margin ratio of 6%, it satisfies the initial margin requirement, but the position is liquidatable because it does not satisfy the maintenance margin of 10%.

Recommendation

We recommend using either the initial margin ratio or maintenance margin ratio across the contract to ensure consistency. Alternatively, we recommend ensuring that the initial margin ratio must be equal to or higher than the maintenance margin ratio.

Status: Resolved

15.Unstable dependencies should not be used in production

Severity: Minor

The dependencies in

- `contracts/margined_fee_pool/Cargo.toml:56`
- `contracts/margined_engine/Cargo.toml:57`
- `contracts/margined_insurance_fund/Cargo.toml:56`
- `contracts/margined_pricefeed/Cargo.toml:55`
- `contracts/margined_vamm/Cargo.toml:57`

are specified in `beta` versions.

Production ready code should not have `beta` dependencies as they could be not fully developed and audited and have a higher likelihood to contain bugs or vulnerabilities.

Recommendation

We recommend updating the mentioned dependencies to the latest stable version.

Status: Resolved

16. Users can open positions with a fractional leverage

Severity: Minor

In `contracts/margined_engine/src/handle.rs:156`, when calculating and then checking the correctness of `margin_ratio`, fractional `leverage` values are allowed.

In fact as `leverage` is expressed in normalized notation, it could represent values smaller than one. That implies that users could open positions with a leverage ratio less than 1, which is not the intended behavior.

Recommendation

We recommend requiring a `leverage` value greater than `config.decimals`.

Status: Resolved

17. Centralization risks

Severity: Minor

The protocols's contracts implement an `Owner` role that has the privilege to run some administrative transactions. As this is an intended behavior, we found that it could lead to centralization risks in the following cases:

- In `contract/margined_fee_pool/src/contract.rs:42`, the `SendToken` message allows the `Owner` to move funds from a `margined_fee_pool` to any arbitrary recipient.
- In `contract/margined_pricefeed/src/contract.rs:43` and `48`, `AppendPrice` and `AppendMultiplePrice` messages allow the `Owner` to set the price of an asset arbitrarily and consequently manipulate the market.

We classify those issues as minor since these centralization risks can only be abused by the owner, who is assumed to be a trusted entity.

Recommendation

We recommend re-thinking the need for those admin functions, and recommend limiting them as much as possible. Additionally, we recommend transfering the admin role to a governance smart contract. Until such a governance contract exists, we recommend setting up a multisig with sufficient signatories, and following operational security best practices, especially regarding key management.

Status: Acknowledged

The team mentioned that this issue is out of scope for the audit work performed. However, as reassurance to the auditors, the team will be using DAO DAO governance for the admin roles within contracts.

18. Lack of configuration parameters validation

Severity: Informational

The contracts within scope lacked validation steps on several configuration parameters upon instantiation or update. This could affect the well-functioning of the protocol if an unexpected value was to be assigned by mistake.

Some parameters are actually checked to be valid ratios within the range between zero and one. However, it is still recommended to perform further validation on those as ratios such as fees very close to '1' or '0' are rarely desired and could greatly affect users.

The below list includes all the affected lines:

- `contracts/margined_engine/src/contract.rs:69,70,71`
- `contracts/margined_engine/src/handle.rs:89,95,101,107`
- `contracts/margined_vamm/src/contract.rs:41-43`
- `contracts/margined_vamm/src/contract.rs:48,49,58,72,73`
- `contracts/margined_vamm/src/handle.rs:47,52,63,69,75,85`

Recommendation

Thorough validation of bounds is recommended on all the configuration parameters of which boundary values could hinder the protocol. In addition, values such as `base_asset_reserve` and `quote_asset_reserve` should be equal to or larger than `decimals` and within a range that allows the proper operation of the vAMM instance.

Status: Partially Resolved

The latest fix only includes checks for `base_asset_reserve` and `quote_asset_reserved`.

19. Lack of pauser role

Severity: Informational

The pause mechanism implemented by the margined engine contract is restricted to the `owner` account as seen in `contracts/margined_engine/src/handle.rs:120`, not following a Role Based Access Controls philosophy. This centralizes all the emergency mechanisms to only one entity.

Recommendation

We recommend implementing a new “pauser” role in charge of the `set_pause` function, improving the granularity of Access Controls. Additionally, for cases where the owner is later expected to be managed by a Governance contract or community, a separate pauser role will allow for swift reactions in case of need, as the account could trigger the pause without waiting for a new governance proposal to pass.

Status: Resolved

20. Sender does not need to be validated

Severity: Informational

In `contracts/margined_engine/src/handle.rs:148` and line `243`, `info.sender` is validated to be a valid address when opening and closing a position. This is inefficient, as Comos SDK already validates `info.sender`.

Recommendation

We recommend not validating the sender's address and directly use `info.sender` instead.

Status: Resolved

21. `validate_address` functionality can be removed

Severity: Informational

In `packages/margined_common/src/validate.rs:48`, the `validate_address` functionality lowercases the provided address value and verifies whether it is equal to the provided address input. As `addr_validate` functionality in Cosmwasm-std now [requires all inputs to be normalized](), this check will be verified automatically when calling `deps.api.addr_validate`.

The same applies to the `addr_validate_to_lower` function located at `packages/margined_common/src/asset.rs:213`.

Recommendation

We recommend removing the `validate_address` and `addr_validate_to_lower` functionalities and directly use `deps.api.addr_validate` instead.

Status: Resolved

22. `require_additional_margin` can be simplified for readability

Severity: Informational

In `contracts/margined_engine/src/utils.rs:256-260`, the `require_additional_margin` functionality attempts to cast unsigned integer into margined common's Integer and verifies whether the position is undercollateralized. This can be simplified into the following code for easier readability:

```
if margin_ratio < base_margin {
      return Err(StdError::generic_err("Position is undercollateralized"));
}
```

A very similar situation was found in the `require_insufficient_margin` function at `contracts/margined_engine/src/utils.rs:270-272`, although the proposed fix would be to use > in this case.

Recommendation

We recommend modifying the `require_additional_margin` and `require_insufficient_margin` function code as described above.

Status: Resolved

23. Confusing error messages and comments

Severity: Informational

In the following instances of the codebase, comments and error messages might negatively impact user experience and readability due to incorrect values:

- The comment in `contracts/margined_fee_pool/src/handle.rs:25` should be `"change owner of fee pool contract"`.
- The error message in `contracts/margined_pricefeed/src/handle.rs:71` should be something like `"Provided prices and timestamps length are not equal"`.

Recommendation

We recommend modifying the comment and error message as mentioned above.

Status: Resolved

24. Multiple inefficient execution flows

Severity: Informational

Multiple execution flows where the main parameters are set to zero and cause no-op operations. These kinds of execution flows are inefficient as they unnecessarily spend gas. The below list details the affected code:

- `contracts/margined_engine/src/handle.rs:133`
    - Executions where `quote_asset_amount` or `leverage` are zero
- `contracts/margined_engine/src/reply.rs`
    - The message on line `574` should be moved to the end of the `if` block of line `567`
- `contracts/margined_fee_pool/src/handle.rs:69`
    - Executions where `amount` is zero

Two instances of incorrectly implemented short-circuits exist in `contracts/margined_vamm/src/handle.rs:300` and `345`. These statements aim to increase the efficiency of the code, but they present incorrect code by returning a value in a Rust function, effectively rendering it ineffective.

Recommendation

Proper short-circuits are recommended for functions that would cause no action or predictable results of zero having an input of zero.

Status: Resolved

25. Outdated and unmaintained dependencies in use

Severity: Informational

The contracts within scope made use of outdated versions of multiple libraries. In addition, the `cosmwasm-bignumber 2.2.0` library depends on `bigint`, which is no longer maintained and affected by the publicly known security vulnerability `CVE-2020-35880`.

We raised this issue as informational given that, at the moment of writting, there are no known exploitation scenarios for `CVE-2020-35880` in CosmWasm smart contracts.

Further details can be found in Appendix-2.

Recommendation

As a general rule, It is recommended to use the latest version of the libraries in scope unless there is a clear reason. This will guarantee that the dependencies' known bugs and vulnerabilities patches are in place.

In the case of `cosmwasm-bignumber 2.2.0`, there is no later version available that does not depend on `bigint`. We recommend updating it to a new version as soon as it becomes available.

Status: Resolved

26. Additional funds sent to the contract are lost

Severity: Informational

In `packages/margined_common/src/asset.rs:75`, a function is implemented to ensure that in the transaction there is a `Coin` with the expected `denom` and `amount` fields.

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

Recommendation

We recommend checking that the transaction contains only the expected `Coin` using https://docs.rs/cw-utils/latest/cw_utils/fn.must_pay.html.

Status: Resolved

27. `CONTRACT_NAME` should include a namespace to be adherent to the `CW2` specification

Severity: Informational

In

- `contracts/margined_fee_pool/src/contract.rs:16`
- `contracts/margined_engine/src/contract.rs:32`
- `contracts/margined_pricefeed/src/contract.rs:15`
- `contracts/margined_insurance_fund/src/contract.rs:17`
- `contracts/margined_vamm/src/contract.rs:22`

the defined `CONTRACT_NAME` values are not including a meaningful namespace as required by the `CW2` specification:

```
"contract is the crate name of the implementing contract, eg.
crate:cw20-base we will use other prefixes for other languages,
and their standard global namespacing"
```

Recommendation

We recommend adding a meaningful namespace to. `CONTRACT_NAME`.

Status: Resolved

Appendix

1. Test case for "Bad debt state is not recorded"

```rust
#[test]
fn test_bad_debt_not_recorded() {

    let SimpleScenario {
        mut router,
        alice,
        engine,
        vamm,
        usdc,
        insurance_fund,
        ..
    } = SimpleScenario::new();

    // insurance contract have 5000 USDC balance
    let insurance_balance = usdc
        .balance::<_, _, Empty>(&router, insurance_fund.addr().clone())
        .unwrap();
    assert_eq!(insurance_balance, to_decimals(5000u64));

    // alice opens a 60 USDC position
    let msg = engine
        .open_position(
            vamm.addr().to_string(),
            Side::Buy,
            to_decimals(60u64),
            to_decimals(10u64),
            to_decimals(0u64),
            vec![],
        )
        .unwrap();
    router.execute(alice.clone(), msg).unwrap();

    // engine contract now have 60 USDC (0 balance + 60)
    let engine_balance = usdc
        .balance::<_, _, Empty>(&router, engine.addr().clone())
        .unwrap();
    assert_eq!(engine_balance, to_decimals(60u64));

    // from oak:
    // we try to trigger bad debt by making the engine contract having
insufficient balance to repay the user
    // we do this via mocking the engine contract to burn all their USDC
    // this would cause the engine contract unable to repay the user, hence
having a positive amount of bad debt
```

```
    // burn all usdc from engine balance
    let burn_msg = Cw20ExecuteMsg::Burn {
        amount: engine_balance,
    };

    // wrap into CosmosMsg
    let cosmos_burn_msg = usdc.call(burn_msg).unwrap();

    // execute the burn msg
    router.execute(engine.addr().clone(), cosmos_burn_msg).unwrap();

    // engine contract now have 0 balance
    let engine_balance = usdc
        .balance::<_, _, Empty>(&router, engine.addr().clone())
        .unwrap();
    assert_eq!(engine_balance, to_decimals(0u64));

    // alice withdraws 20 USDC margin from engine contract while engine contract
have 0 balance
    let msg = engine
        .withdraw_margin(vamm.addr().to_string(), to_decimals(20u64))
        .unwrap();
    router.execute(alice.clone(), msg).unwrap();

    // shortfall event occured, insurance funds are used instead
    // insurance contract should have 4980 USDC (5000 balance - 20)
    let insurance_balance = usdc
        .balance::<_, _, Empty>(&router, insurance_fund.addr().clone())
        .unwrap();
    assert_eq!(insurance_balance, to_decimals(4980u64));

    // engine contract's state should reflect bad debt as 20
    let bad_debt = engine.state(&router).unwrap().bad_debt;
    assert_eq!(bad_debt, to_decimals(20u64));

}
```

2. Outdated dependencies

List of Outdated dependencies.

```
Name             Project  Compat  Latest   Kind    Platform
----             -------  ------  ------   ----    --------
cw-utils         0.13.4   ---     0.14.0   Normal  ---
cw2               0.13.4   ---     0.14.0   Normal  ---
cw20              0.13.4   ---     0.14.0   Normal  ---
cw20-base         0.13.4   ---     0.14.0   Normal  ---
serde            1.0.138  ---     1.0.143  Normal  ---
serde_derive     1.0.138  ---     1.0.143  Normal  ---
serde_json       1.0.82   ---     1.0.83   Normal  ---
serde-json-wasm   0.3.2    ---     0.4.1    Normal       ---
thiserror        1.0.31   ---     1.0.32   Normal  ---
thiserror-impl   1.0.31   ---     1.0.32   Normal  ---
strum_macros     0.24.2   ---     0.24.3   Normal  ---
cw-multi-test     0.13.4   ---     0.14.0   Normal  ---
cw-storage-plus   0.13.4   ---     0.14.0   Normal  ---
sha3              0.10.1   ---     0.10.2   Normal       ---
```

Information related to the unmaintained bigint library.

```
Crate:     bigint
Version:   4.4.3
Warning:   unmaintained
Title:     bigint is unmaintained, use uint instead
Date:      2020-05-07
ID:        RUSTSEC-2020-0025
URL:       https://rustsec.org/advisories/RUSTSEC-2020-0025
Dependency tree:
bigint 4.4.3
└── cosmwasm-bignumber 2.2.0
    ├── margined_vamm 0.1.0
    │   ├── margined_utils 0.1.0
    │   │   ├── margined_vamm 0.1.0
    │   │   ├── margined_insurance_fund 0.1.0
    │   │   │   └── margined_utils 0.1.0
    │   │   ├── margined_fee_pool 0.1.0
    │   │   │   └── margined_utils 0.1.0
    │   │   └── margined_engine 0.1.0
    │   │       ├── margined_vamm 0.1.0
    │   │       └── margined_utils 0.1.0
    │   └── margined_engine 0.1.0
    ├── margined_utils 0.1.0
    ├── margined_pricefeed 0.1.0
    │   ├── margined_vamm 0.1.0
│   └── margined_utils 0.1.0
├── margined_perp 0.1.0
│   ├── margined_vamm 0.1.0
│   ├── margined_utils 0.1.0
│   ├── margined_pricefeed 0.1.0
│   ├── margined_insurance_fund 0.1.0
│   ├── margined_fee_pool 0.1.0
│   └── margined_engine 0.1.0
├── margined_insurance_fund 0.1.0
├── margined_fee_pool 0.1.0
├── margined_engine 0.1.0
└── margined_common 0.1.0
    ├── margined_vamm 0.1.0
    ├── margined_utils 0.1.0
    ├── margined_pricefeed 0.1.0
    ├── margined_perp 0.1.0
    ├── margined_insurance_fund 0.1.0
    ├── margined_fee_pool 0.1.0
    └── margined_engine 0.1.0
```

3. Test case for “Incorrect accounting during liquidation”

```rust
#[test]
fn test_liquidation() {
    let SimpleScenario {
        mut router,
        alice,
        bob: _,
        carol,
        owner,
        engine,
        usdc,
        vamm,
        pricefeed,
        insurance_fund,
        ..
    } = SimpleScenario::new();

    // set the latest price
    let price: Uint128 = Uint128::from(10_000_000_000u128);
    let timestamp: u64 = router.block_info().time.seconds();

    let msg = pricefeed
        .append_price("ETH".to_string(), price, timestamp)
        .unwrap();
    router.execute(owner.clone(), msg).unwrap();

    router.update_block(|block| {
        block.time = block.time.plus_seconds(900);
        block.height += 1;
    });

    // set maintenance ratio as 10% to allow liquidation
    let msg = engine
        .set_maintenance_margin_ratio(Uint128::from(100_000_000u128))
        .unwrap();
    router.execute(owner.clone(), msg).unwrap();

    // set max liquidation fee to trigger shortfall event
    let msg = engine
        .set_liquidation_fee(to_decimals(1u64))
        .unwrap();
    router.execute(owner.clone(), msg).unwrap();

    /*
        Pre liquidate
    */
    // engine contract balance: 0 USDC
    let engine_balance = usdc
```

```
      .balance::<_, _, Empty>(&router, engine.addr().clone())
      .unwrap();
     assert_eq!(engine_balance, Uint128::from(0u128));

     // alice creates 25 USDC position
     let msg = engine
         .open_position(
             vamm.addr().to_string(),
             Side::Buy,
             to_decimals(25u64),
             to_decimals(10u64),
             to_decimals(0u64),
             vec![],
         )
         .unwrap();
     router.execute(alice.clone(), msg).unwrap();

     router.update_block(|block| {
         block.time = block.time.plus_seconds(15);
         block.height += 1;
     });

     // insurance contract balance: 5000 USDC
     let insurance_balance = usdc
         .balance::<_, _, Empty>(&router, insurance_fund.addr().clone())
         .unwrap();
     assert_eq!(insurance_balance, Uint128::from(to_decimals(5000u64)));

     // query engine contract balance to make sure 25 USDC exist
     let engine_balance = usdc
     .balance::<_, _, Empty>(&router, engine.addr().clone())
     .unwrap();
    assert_eq!(engine_balance, to_decimals(25u64));

    // attempt liquidation
     let msg = engine
         .liquidate(
             vamm.addr().to_string(),
             alice.to_string(),
             to_decimals(0u64),
         )
         .unwrap();
     let liquidation_attribute = router.execute(carol.clone(), msg).unwrap();
     assert_eq!(liquidation_attribute.events[5].attributes[2].value,
to_decimals(125u64).to_string());

     /*
         Post liquidate
     */
     // Engine contract balance should be 0
     let engine_balance = usdc
         .balance::<_, _, Empty>(&router, engine.addr().clone())
         .unwrap();
    assert_eq!(engine_balance, to_decimals(0u64));

    // insurance contract balance, 5000-100 = 4900
     let insurance_balance = usdc
         .balance::<_, _, Empty>(&router, insurance_fund.addr().clone())
         .unwrap();
     assert_eq!(insurance_balance, to_decimals(4900u64));

     // liquidator (carol) balance should have 125 USDC
     let liquidator_balance = usdc
         .balance::<_, _, Empty>(&router, carol.clone())
         .unwrap();
     assert_eq!(liquidator_balance, to_decimals(125u64));
 }
```

4. Test case for “Low decimals configured in virtual AMM might cause incorrect swap returns”

```rust
#[test]
fn swap_input_small_amount() {
    // reproduced in contracts/margined_vamm/src/testing/swap_tests.rs
    let mut deps = mock_dependencies();
    let msg = InstantiateMsg {
        decimals: 0u8, // low decimals configured
        quote_asset: "USDT".to_string(),
        base_asset: "BTC".to_string(),
        quote_asset_reserve: Uint128::from(12_000u64), // 1 BTC = 1200 USDT
        base_asset_reserve: Uint128::from(10_u64),
        funding_period: 3_600_u64,
        toll_ratio: Uint128::zero(),
        spread_ratio: Uint128::zero(),
        fluctuation_limit_ratio: Uint128::zero(),
        pricefeed: "oracle".to_string(),
        margin_engine: Some("addr0000".to_string()),
    };
    let info = mock_info("addr0000", &[]);
    instantiate(deps.as_mut(), mock_env(), info, msg).unwrap();

    // decimals auto configured as 1
    let res = query(deps.as_ref(), mock_env(), QueryMsg::Config {}).unwrap();
    let config: ConfigResponse = from_binary(&res).unwrap();
    assert_eq!(config.decimals, Uint128::from(1u64));

    // open amm
    let info = mock_info("addr0000", &[]);
    execute(
        deps.as_mut(),
        mock_env(),
        info,
        ExecuteMsg::SetOpen { open: true },
    )
    .unwrap();

    // Swap in USD
    let swap_msg = ExecuteMsg::SwapInput {
        direction: Direction::RemoveFromAmm,
        quote_asset_amount: Uint128::from(1u64), // remove 1 USDT, should
increase BTC price by 0.000833333333, but rounded down to 0
        base_asset_limit: Uint128::zero(),
        can_go_over_fluctuation: false,
    };

    let info = mock_info("addr0000", &[]);
    let input_res = execute(deps.as_mut(), mock_env(), info, swap_msg).unwrap();
    assert_eq!(input_res.attributes[6].value, "1".to_string()); //
quote_asset_amount
    assert_eq!(input_res.attributes[7].value, "1".to_string()); //
base_asset_amount

    let res = query(deps.as_ref(), mock_env(), QueryMsg::State {}).unwrap();
    let state: StateResponse = from_binary(&res).unwrap();
    assert_eq!(
        state,
        StateResponse {
            open: true,
            quote_asset_reserve: Uint128::from(11_999u64), // 12_000 - 1 =
11_999
            base_asset_reserve: Uint128::from(11u64), // but BTC increased by 1
instead of 10.000833333333, due to contracts/margined_vamm/src/handle.rs:328
            total_position_size: Integer::new_negative(1u64),
            funding_rate: Integer::zero(),
            next_funding_time: 1_571_801_019u64,
        }
    );
}
```

5. **Test case for “Incorrect comparison of negative values on the Integer library”**

```
#[test]
fn incorrect_negative_comparison()  {
    let a = Integer::new_negative(1u64);
    let b = Integer::new_negative(2u64);

    assert_eq!(false, a > b);
    assert_eq!(false, a.cmp(&b) == Ordering::Greater);
    assert_eq!(false, a.partial_cmp(&b) == Some(Ordering::Greater));

}
```