NEAR Burrow

Burrowland - Smart Contracts

Security Assessment Report

Version: 2.0

October, 2024

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the NEAR Burrow smart contracts. The review focused solely on the security aspects of the Rust implementation of the contracts, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the NEAR Burrow smart contracts contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the NEAR Burrow smart contracts.

Overview

Burrowland is a decentralised lending platform built on the NEAR blockchain. It enables users to lend and borrow assets while earning interest on their deposits.

This review primarily focused on margin trading functionality, which aims to allow users to efficiently open margin positions with just one operation, while benefiting from all the existing liquidity of the Burrow platform.

Security Assessment Summary

Scope

The review was conducted on the files hosted on the burrowland repository.

The scope of this time-boxed review was strictly limited to files at commit c09a41b. Retesting activities were performed on commit fb484de.

Note: third party libraries and dependencies, were excluded from the scope of this assessment.

Approach

The manual review focused on identifying issues associated with the business logic implementation of the contracts. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the NEAR Blockchain.

To support this review, the testing team also utilised the following automated testing tools:

- `cargo audit` : `https://crates.io/crates/cargo-audit`
- `cargo deny` : `https://github.com/EmbarkStudios/cargo-deny`
- `cargo tarpaulin` : `https://crates.io/crates/cargo-tarpaulin`
- `cargo geiger` : `https://github.com/rust-secure-code/cargo-geiger`
- `clippy` : `https://github.com/rust-lang/rust-clippy`

Output for these automated tools is available upon request.

Coverage Limitations

Due to the time-boxed nature of this review, all documented vulnerabilities reflect best effort within the allotted, limited engagement time. As such, Sigma Prime recommends to further investigate areas of the code, and any related functionality, where majority of critical and high risk vulnerabilities were identified.

Findings Summary

The testing team identified a total of 22 issues during this assessment. Categorised by their severity:

- High: 5 issues.
- Medium: 5 issues.
- Low: 4 issues.
- Informational: 8 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the NEAR Burrow smart contracts. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as “informational”.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
BUR-01	Margin Withdraw Function Lacks Critical Checks	High	Resolved
BUR-02	Lack Of Minimum Debt Requirement Causes Bad Debt	High	Resolved
BUR-03	Unchecked Panic With `unwrap()` And `expect()`	High	Closed
BUR-04	Untracked Storage Usage For Margin Accounts	High	Resolved
BUR-05	Liquidator-Controlled Minimum Swap Amount In Liquidations And Force Closures	High	Closed
BUR-06	Block Timestamps Manipulation	Medium	Closed
BUR-07	`force_closing_enabled` Is Not Checked For Margin Positions	Medium	Resolved
BUR-08	Incorrect Calculations Lead To Higher Holding Position Fee	Medium	Resolved
BUR-09	Stale Debt Cap Used In Holding Position Fee Calculation	Medium	Closed
BUR-10	Inefficient Hashmap Usage In Contract State	Medium	Closed
BUR-11	Default Prices May Deviate From Market Prices	Low	Closed
BUR-12	Lack Of Minimum Shares Check For Margin Debt	Low	Resolved
BUR-13	No Health Check When Decreasing A Margin Position	Low	Closed
BUR-14	Missing Pyth Oracle Enablement Check For Margin Actions	Low	Resolved
BUR-15	All Leftover Funds Are Sent To Liquidator After Liquidation	Informational	Resolved
BUR-16	Permissionless Pyth Price Updates Allow For Micro Price Manipulations	Informational	Closed
BUR-17	Configuration Options In `AssetConfig` Are Unused By Margin Actions	Informational	Resolved
BUR-18	`u64` And `u128` Types Are Used As Inputs And Outputs	Informational	Closed
BUR-19	Use Of Deprecated Pyth Oracle `get_price()` Function	Informational	Resolved
BUR-20	Unused Pyth Best Practice Confidence Intervals In Price Calculations	Informational	Closed
BUR-21	Potential For Storage Abuse In Callbacks And Trusted Functions	Informational	Closed
BUR-22	Miscellaneous General Comments	Informational	Resolved

BUR-01	Margin Withdraw Function Lacks Critical Checks		
Asset	`margin_actions.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description

The margin withdraw function `internal_margin_withdraw_supply()` misses several checks compared to the normal withdraw function `internal_withdraw()`:

- It does not check if the amount to withdraw exceeds the available amount `asset.available_amount()` for this asset. This may lead to parts of the reserves or protocol fees being withdrawn, resulting in DoS issues.
- It does not check check that `asset.config.can_withdraw` is `true`. This means that assets that are not normally allowed to be withdrawn can still be withdrawn through the margin withdraw functions, circumventing withdrawal restrictions.
- It does not check if the token to withdraw is a shadow token. However, no direct impact for this absent check was identified during testing.

Recommendations

Implement missing checks on the `internal_margin_withdraw_supply()` function.

Consider simplifying and refactoring the implementation so that there is only one withdraw function.

Resolution

This issue has been addressed in commit fb484de by adding the missing checks. The development team added the following response:

> *Fixed. Since regular positions and margin positions are under different accounts, and the deposit and collateral operations are also separate, there are no plans to merge the withdraw function for now.*

BUR-02	Lack Of Minimum Debt Requirement Causes Bad Debt		
Asset	`actions.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description

The contract currently does not enforce a minimum debt amount during borrowing, liquidation or repayment. This applies to both regular positions and margin positions.

The `internal_borrow()` function below illustrates the lack of any checks for a minimum debt amount:

actions.rs

```
pub fn internal_borrow(
    &mut self,
    position: &String,
    account: &mut Account,
    asset_amount: &AssetAmount,
) -> Balance {
    // ...

    // No checks for minimum debt amount

    asset.borrowed.deposit(borrowed_shares, amount);
    asset.supplied.deposit(supplied_shares, amount);
    self.internal_set_asset(&asset_amount.token_id, asset);

    account.increase_borrowed(position, &asset_amount.token_id, borrowed_shares);

    // ...
}
```

This omission opens the door to potential dust attacks, where an attacker can create many small debt positions. These small debts can become unprofitable to liquidate, causing the protocol to accumulate small amounts of bad debt indefinitely.

Recommendations

Consider implementing a minimum debt requirement for both normal and margin positions. Ensure this is checked when opening and decreasing a position for consistency across all debt-related operations.

The minimum debt amount should be set high enough such that any debt position remains economically viable for liquidation.

Resolution

This issue has been fixed for margin accounts in commit fb484de by setting a minimum debt amount. The development team added the following comment:

Additional logic has been added to prevent the occurrence of dust in the opening and closing logic of margin positions: When opening a position, the amount of debt tokens is checked. When closing a position, it is verified that repaying the debt with the `min_token_d_amount` *will not leave any dust. However, it is still possible for the position's debt to become dust due to swap results, but dust is not allowed to continue decreasing. Additionally, after the modification, both regular positions and margin positions will incur storage fees, giving users the incentive to clear dust debt positions.*

No mitigation was implemented for regular accounts with the following reasoning:

1. Users have already paid storage fees for dust debt, so they have the incentive to clear the dust, which has a minimal impact on the protocol.
2. Adding restrictions to the Borrow and Repay commands would significantly impact liquidation bots, as some of the current dust debts are generated from liquidations.

BUR-03	Unchecked Panic With `unwrap()` And `expect()`		
Asset	`account.rs`		
Status	**Closed:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

It was observed that multiple instances of the `unwrap()` and `expect()` methods are used throughout the code. These functions, while convenient for handling `Option` and `Result` types, can lead to unhandled panics if called on a `None` value or an `Err` variant without sufficient validation.

Panics triggered by `unwrap()` or `expect()` can terminate the program abruptly, potentially leading to unexpected behaviour or denial of service, particularly in production systems where stability and fault tolerance are critical.

Recommendations

Handle `Option` and `Result` types more robustly by using safer alternatives such as:

1. **Pattern matching**: Explicitly handle both `Some` / `None` and `Ok` / `Err` cases using `match` statements, ensuring all possible cases are covered.
2. **`if let` or `while let` constructs**: These provide more concise handling of successful cases, with fallback behaviour in case of errors or missing values.
3. **Custom error handling**: Return relevant error messages or propagate errors using the `?` operator to gracefully handle failure scenarios, allowing errors to propagate up to higher levels of the application.

Resolution

The development team has acknowledged these comments and closed the issue with the following response:

> *We have checked the places where `unwrap()` and `expect()` are used, and it is confirmed that state rollbacks are necessary in these places.*

BUR-04	Untracked Storage Usage For Margin Accounts		
Asset	`margin_account.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description

The `internal_set_margin_account()` function in the `margin_account` implementation does not track storage usage when inserting or updating margin accounts.

This is inconsistent with the handling of regular accounts, where storage is typically tracked and charged to users.

margin_account.rs

```
84 pub(crate) fn internal_set_margin_account(&mut self, account_id: &AccountId, account: MarginAccount) {
       self.margin_accounts.insert(account_id, &account.into());
86 }
```

This oversight may lead to users locking up NEAR in the contract without paying for the associated storage costs.

Malicious actors could create numerous margin accounts or large accounts, consuming contract storage without cost, causing Denial-of-Service (DoS) conditions.

Recommendations

Implement storage tracking for margin accounts, similar to the one on regular accounts.

Resolution

This issue has been addressed in commit fb484de by adding storage tracking for margin accounts.

BUR-05	Liquidator-Controlled Minimum Swap Amount In Liquidations And Force Closures		
Asset	`margin_position.rs`		
Status	**Closed:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

In the `process_decrease_margin_position()` function, liquidators can manipulate the `min_token_d_amount` parameter during liquidations or forced closures. This parameter controls the minimum token amount that must be returned from a DEX swap, and its improper handling can lead to value extraction.

The lower bound of `min_token_d_amount` is checked by the `is_min_amount_out_reasonable()` function, which uses current oracle prices and the `max_slippage_rate`. However, the reliance on these inputs allows liquidators to exploit the system by setting this parameter in a way that benefits them, potentially leading to financial losses for the protocol:

1. Liquidators can set `min_token_d_amount` to the lowest value within the allowed range.
2. They can then sandwich the trade themselves, extracting value up to the `max_slippage_rate`.

This can lead to two adverse outcomes:

- For liquidations: Less funds remain after liquidation, effectively taking value from the position owner.
- For force closures: More bad debt accumulates, taking value from the protocol and impacting its solvency.

Recommendations

Implement stricter validation mechanisms for `min_token_d_amount` to ensure it aligns with market conditions and prevents manipulation.

Set `max_slippage_rate` to a low value such that the potential value extraction is negligible, while allowing for some slippage so that liquidations can go through in time.

Resolution

The development team has acknowledged these comments and responded with the following rationale:

> *`is_min_amount_out_reasonable()` is a function used to check whether `min_token_d_amount` is consistent with the market. `max_slippage_rate` will be set to an appropriate value.*

BUR-06	Block Timestamps Manipulation		
Asset	`account.rs`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

The `sync_booster_policy()` function, which relies on block timestamps to calculate staking durations, presents a potential vulnerability where if block timestamps are not properly synchronized or controlled, time manipulation could occur.

This vulnerability could allow attackers to artificially extend or shorten staking durations, leading to unintended or unfair booster token staking behaviour.

Recommendations

Ensure accurate time-based operations by relying on consistent timestamp sources and validating them against possible inconsistencies.

Implement additional checks to ensure staking logic is not exploitable by manipulating block timestamps.

Resolution

The development team has acknowledged these comments and responded with the following rationale:

> `env::block_timestamp` *is the most reliable time source available.*

BUR-07	`force_closing_enabled` Is Not Checked For Margin Positions		
Asset	`config.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

`force_closing_enabled` is not checked when force-closing a margin position.

A position can be *"force-closed"* if it has accrued bad debt. In such cases, the protocol reserves will be used to cover that debt. `force_closing_enabled` is a configuration option that enables or disables force-closing of positions.

`force_closing_enabled` is only checked when force-closing a normal position, but it is not checked for margin positions. As such, margin positions can still be force-closed even when `force_closing_enabled` is `false`.

This means that there is no way to stop the protocol reserves from being used to cover bad debt on margin positions.

Recommendations

Implementing a check for `force_closing_enabled` when force closing a margin position, similar to a normal position.

Resolution

This issue has been addressed in commit fb484de by ensuring `force_closing_enabled` is true when force closing a margin position.

BUR-08	Incorrect Calculations Lead To Higher Holding Position Fee		
Asset	`margin_trading.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Low	Likelihood: High

Description

When decreasing a margin position, the 'holding position fee' is calculated using the difference between the `uahpi_at_open` and the current `uahpi` of that asset:

margin_trading.rs

```
409 // ...
    let hp_fee = u128_ratio(
411   mt.debt_cap,
      asset_debt.unit_acc_hp_interest - mt.uahpi_at_open,
413   UNIT,
      );
415
    // mt.uahpi_at_open is not updated
```

However, after the `hp_fee` is subtracted from the users' balance, `uahpi_at_open` is not updated to the current `uahpi`.

This means that when decreasing the position a second time, the `hp_fee` will have to be paid again for the entire period (from open to current), resulting in duplicate fees.

Recommendations

Refactor the code to update `uahpi_at_open` when the `hp_fee` is paid.

Resolution

This issue has been addressed in commit fb484de by updating `uahpi_at_open` appropriately.

BUR-09	Stale Debt Cap Used In Holding Position Fee Calculation		
Asset	`margin_trading.rs`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: Low	Likelihood: High

Description

In the `on_decrease_trade_return()` function, the holding position fee (`hp_fee`) is calculated using `mt.debt_cap`, which represents the initial debt amount when the position was opened. However, this value is not updated over time to reflect accrued interest. This results in an underestimation of the actual fee:

margin_trading.rs

```
410 let hp_fee = u128_ratio(
        mt.debt_cap,
412     asset_debt.unit_acc_hp_interest - mt.uahpi_at_open,
        UNIT,
414 );
```

The actual debt amount increases over time due to interest accrual, but `mt.debt_cap` remains unchanged. This results in `hp_fee` being calculated based on the initial debt amount, instead of the current debt balance, thus making `hp_fee` smaller than its actual value.

Recommendations

Remove the use of `mt.debt_cap` and instead, calculate the current debt amount every time it is needed.

Resolution

The development team has acknowledged these comments and decided to close the issue with the following comment:

> *This is according to product requirements.*

BUR-10	Inefficient Hashmap Usage In Contract State		
Asset	`lib.rs`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: Low	Likelihood: High

Description

Several data structures such as `last_prices`, `last_lp_token_infos`, `token_pyth_info`, `last_staking_token_prices`, `Account`, `AccountFarm`, and `MarginAccount` use `std::collections::HashMap` for storing state information.

Using `std::collections::HashMap` can lead to significant performance issues and potential security vulnerabilities. Each time a function with `self` is called, the entire `HashMap` is loaded into memory, regardless of whether all the data is needed.

Loading unnecessary data consumes more gas, making contract interactions more expensive. Moreover, if the `HashMap` becomes too large, it could lead to Denial-of-Service (DoS) conditions.

Recommendations

Replace `std::collections::HashMap` with relevant data structures from `near_sdk::collections`, which were designed to be more efficient for `NEAR` smart contracts and will only load data on demand.

Resolution

The development team has acknowledged these comments and decided to close the issue with the following rationale:

> *Currently, the root storage that uses the `HashMap` type is for small amounts of data. When the data volume needs to increase in the future, it will be changed to `near_sdk::collections`.*

BUR-11	Default Prices May Deviate From Market Prices		
Asset	`pyth.rs`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

The protocol allows the owner to set a "default price" for an asset, which is then used for valuation instead of the price fetched from an oracle. This creates a risk where the default price may diverge significantly from the real market price. Such discrepancy could result in overvalued collateral and undervalued debt, potentially leading to the accrual of bad debt and value extraction from the protocol.

Therefore, setting a default price for an asset requires careful consideration. For instance, if a stablecoin experiences a significant depeg while a default price is in place, it could be exploited as collateral, potentially draining the entire protocol.

Recommendations

Ensure the above comments are understood.

To mitigate the risks associated with setting a default price for an asset, consider implementing the following measures:

1. **Implement Safeguards**: Introduce strict controls and validation checks when setting or updating the default price, ensuring it closely aligns with the current market price. This could include limits on how much the default price can deviate from the oracle price.
2. **Dynamic Pricing Mechanism**: Consider using a dynamic pricing mechanism that automatically adjusts the default price based on predefined rules or market conditions, reducing reliance on manual price settings.
3. **Emergency Procedures**: Establish emergency procedures that can quickly disable or adjust the default price in response to market volatility, such as a stablecoin depeg, to prevent potential exploitation.

Resolution

The development team has acknowledged these comments and decided to close the issue with the following rationale:

> *The default price will only be used for unlisted tokens, such as USN, that do not have a price.*

BUR-12	Lack Of Minimum Shares Check For Margin Debt		
Asset	`asset.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

In the `internal_set_asset()` function, there are checks such that `supplied` and `borrowed` shares meet a minimum threshold. However, there are no similar checks for `asset.margin_debt.shares`:

asset.rs

```
230 pub fn internal_set_asset(&mut self, token_id: &TokenId, mut asset: Asset) {
        assert!(asset.supplied.shares.0 == 0 ||
232         asset.supplied.shares.0 >= MIN_RESERVE_SHARES, "Asset {} supply shares cannot be less than {}", token_id,
                ↪ MIN_RESERVE_SHARES);
        assert!(asset.borrowed.shares.0 == 0 ||
234         asset.borrowed.shares.0 >= MIN_RESERVE_SHARES, "Asset {} borrow shares cannot be less than {}", token_id,
                ↪ MIN_RESERVE_SHARES);
        // ...
236 }
```

This omission opens up possibilities for inflation attacks. For example, an attacker may be able to inflate away their debt.

Recommendations

Implement a minimum shares check for `asset.margin_debt`, similar to the ones for `supplied` and `borrowed`.

Resolution

This issue has been addressed in commit fb484de by implementing a minimum shares check for `asset.margin_debt`.

BUR-13	No Health Check When Decreasing A Margin Position		
Asset	`margin_position.rs`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

When a margin position is decreased, there are no health checks on the decreased position. This means that the position may become liquidatable as a result of a decrease action.

This can occur when the user sets `min_token_d_amount` too low and the swap has lots of slippage.

In the worst case, the position may even end up being applicable to force-closing, meaning that bad debt has accrued and the protocol has lost funds. However, this would require `min_token_d_amount` to be set very low and considering that it is currently limited by `is_min_amount_out_reasonable()`, this seems very unlikely.

Recommendations

Add a health check in `process_decrease_margin_position()` to ensure that the position would not be applicable for liquidation or force-closing even for the worst case outcome of the swap.

Resolution

The development team has acknowledged these comments and decided to close the issue with the following rationale:

> *This issue will be avoided by setting a reasonable* `max_slippage_rate`.

BUR-14	Missing Pyth Oracle Enablement Check For Margin Actions		
Asset	`margin_pyth.rs`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

In the `internal_margin_execute_with_pyth()` function, the `self.internal_config().enable_pyth_oracle` flag is not checked to confirm whether the Pyth oracle is enabled before executing margin actions. This differs from non-margin actions, where the check is properly implemented.

As a result, the Pyth oracle might be used unintentionally during margin operations, potentially leading to unintended behaviour or reliance on the oracle when it is not intended to be active.

Recommendations

Add a check in the `internal_margin_execute_with_pyth()` function to make sure that the Pyth oracle is enabled.

Resolution

This issue has been addressed in commit fb484de by adding a check for `self.internal_config().enable_pyth_oracle`.

BUR-15	All Leftover Funds Are Sent To Liquidator After Liquidation
Asset	`margin_trading.rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

After a liquidation is complete, all leftover funds are sent to the liquidator, not the owner of the liquidated position.

This seems like a harsh penalty to the owner of the position. In general, it is common for the liquidator to take a small fee to cover their operational costs, but most of the leftover funds are sent back to the position owner.

Recommendations

Ensure the above is understood and consider changes if different approach of handling leftover funds is deemed necessary.

Resolution

This issue has been addressed in commit fb484de by distributing the remaining tokens proportionally among the user, the protocol, and the liquidator.

BUR-16	Permissionless Pyth Price Updates Allow For Micro Price Manipulations
Asset	`pyth.rs`
Status	**Closed:** See Resolution
Rating	Informational

Description

The permissionless nature of Pyth price updates introduces a potential issue where users can selectively publish or withhold price updates to manipulate outcomes in their favour. This selective publishing could be leveraged to exploit other parts of the system.

Pyth price updates are permissionless, allowing any user to update prices onchain. Users listen to the offchain Pyth price feed, which broadcasts new signed prices multiple times per second. They can then choose to publish one of these prices to the Pyth smart contracts, after which any smart contract querying the onchain price will receive the updated value.

This system allows users to selectively pick and publish price updates or choose not to publish them, depending on what benefits them most. While this behaviour isn't inherently problematic, it could be exploited as part of broader attack strategies.

For example, a malicious liquidator, as per finding BUR-05, could publish a price update that is 0.5% lower than the current oracle price. This would result in `is_min_amount_out_reasonable()` being 0.5% lower, allowing the liquidator to extract 0.5% more profit to the detriment of the protocol.

This issue is exacerbated during high volatility times.

Recommendations

Ensure the above comments are understood. Seeing as permissionless price updates are inherent to Pyth, there is no clear mitigation.

Resolution

The development team has acknowledged these comments and decided to close the issue with the following rationale:

> *We have our own Pyth price reporting bot that continuously reports prices.*

BUR-17	Configuration Options In `AssetConfig` Are Unused By Margin Actions
Asset	`asset_config.rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

Several configuration options in `AssetConfig` are not used or checked by margin actions:

- `can_use_as_collateral` is not used. This means that there is no control on which assets can be used as collateral or position tokens in margin positions.
- `can_borrow` is not used. This means there is no control on which assets are used as debt tokens in margin positions.
- `can_withdraw` is not used. This is discussed in finding BUR-01

Recommendations

Consider if it is desirable to have these configurations enabled for margin actions and implement relevant checks if deemed applicable.

Resolution

This issue has been addressed in commit fb484de by adding the appropriate checks.

BUR-18	`u64` And `u128` Types Are Used As Inputs And Outputs
Asset	`config.rs`
Status	**Closed:** See Resolution
Rating	Informational

Description

It is best practice to use SDK types `U64` and `U128` instead of native types `u64` and `u128` as inputs and outputs for functions.

This is because large numeric values like `u64` and `u128` cannot be directly represented in JSON, the default encoding format for function inputs and outputs.

`u64` and `u128` are used in the following places:

- `config.rs` line [**158**]
- `account.rs` line [**262**]
- `margin_accounts.rs` line [**173**]

Due to no direct impact identified during the allotted time to this assessment, this finding is rated as informational only.

Recommendations

Ensure the above comments are understood and consider replacing the native types with the SDK types.

Resolution

The development team has acknowledged these comments and decided to close the issue with the following rationale:

> *The business logic ensures that the input parameters will not exceed* 2^{53}

BUR-19	Use Of Deprecated Pyth Oracle `get_price()` Function
Asset	`pyth.rs`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The contract is currently using the deprecated `getPrice()` function from the Pyth oracle.

According to the Pyth documentation, this function should be replaced with `getPriceNoOlderThan()`.

Recommendations

Replace `getPrice()` with `getPriceNoOlderThan()` as per Pyth advisory.

Resolution

This issue has been addressed in commit fb484de by using `getPriceNoOlderThan()`.

BUR-20	Unused Pyth Best Practice Confidence Intervals In Price Calculations
Asset	`pyth.rs`
Status	**Closed:** See Resolution
Rating	Informational

Description

The contract currently uses price data from the Pyth oracle, but does not utilize the confidence intervals provided alongside the current price.

Pyth offers these confidence intervals as a measure of price uncertainty, which could be particularly valuable during periods of high market volatility.

Recommendations

Consider implementing a more nuanced pricing strategy that incorporates Pyth's confidence intervals:

1. **For collateral valuation**: consider using the lower bound of the confidence interval. This would result in a more conservative estimate of collateral value.
2. **For debt valuation**: consider using the upper bound of the confidence interval. This would result in a more conservative estimate of debt owed.

Resolution

The development team has acknowledged these comments and decided to close the issue with the following rationale:

> *An asset already has volatility, and the other oracle does not have a confidence interval, so we will uniformly use volatility.*

BUR-21	Potential For Storage Abuse In Callbacks And Trusted Functions
Asset	`shadow_actions.rs`
Status	**Closed:** See Resolution
Rating	Informational

Description

Two areas of the contract have been identified where trusted entities could potentially abuse storage, causing Denial-of-Service (DoS) conditions:

1. In the `callback_sync_lp_infos()` function, the contract writes the contents of the promise result from the ref exchange to `self.last_lp_token_infos` in storage. If the ref exchange includes a large amount of data in the promise result, it could cause excessive data to be written to storage.
2. In the `extend_blacklist_of_farmers()` function, guardians can write data to `self.blacklist_of_farmers`. There is no limit on the amount of data that can be written, potentially allowing guardians to fill storage with a large number of blacklisted farmers.

While these issues are mitigated by the fact that both the ref exchange and guardians are trusted entities, they represent a potential vector for unintended storage growth or DoS attacks if these trusted entities are compromised or act maliciously.

Recommendations

Monitor the growth of these storage areas and implement additional safeguards if necessary.

Resolution

The development team has acknowledged these comments and decided to close the issue

BUR-22	Miscellaneous General Comments
Asset	All contracts
Status	**Resolved:** See Resolution
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **Incomplete Balance Check When Changing Asset Extra Decimals**

   Related Asset(s): config.rs

   In the `update_asset()` function, when changing the `extra_decimals` of an asset, the contract checks if certain balances are zero. However, this check is incomplete as it does not account for all possible balances, namely the margin balances.

   While no immediate negative impact is apparent, this inconsistency should be corrected.

2. **Insufficient And Low-Quality Code Documentation**

   Related Asset(s): *.rs

   Throughout the codebase, there are numerous instances of missing, insufficient, or poorly written comments and documentation. Existing comments are often incomplete, or contain spelling errors. There is also inconsistent use of documentation styles across the codebase.

   There are also multiple examples of comments that still contain `TODO` statements. This makes it unclear whether this feature should be present or not.

   Review the entire codebase and improve comments and documentation where necessary. Consider implementing consistent use of NatSpec. Consider using a spell-checker to correct any spelling errors in existing comments.

   Consider either removing the `TODO` or implementing the features mentioned.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The development team has acknowledged these comments and addressed some of the above issues in commit fb484de.

Appendix A Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

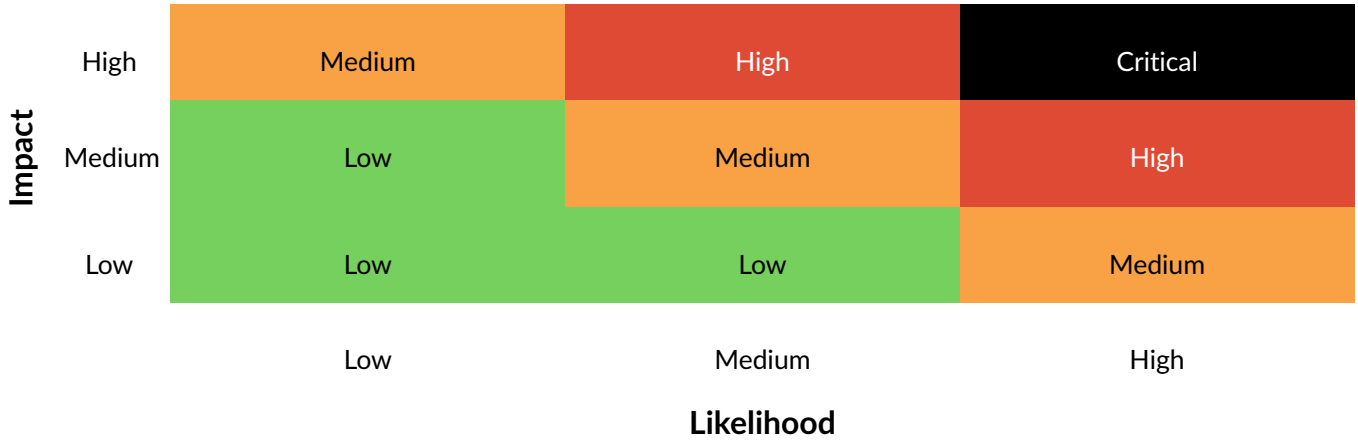

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.