

# Security Audit Report

---

## Templar Protocol Templar Smart Contracts

Initial Report // April 30th, 2025

Final Report // July 1st, 2025

### Team Members

Ahmad Jawid Jamiulahmadi // Senior Security Auditor

Mukesh Jaiswal // Senior Security Auditor



# Table of Contents

<u>1.0 Scope</u>	4
<u>1.1 Technical Scope</u>	
<u>1.2 Documentation</u>	
<u>2.0 Executive Summary</u>	5
<u>2.1 Schedule</u>	
<u>2.2 Overview</u>	
<u>2.3 Threat Model</u>	
<u>2.4 Security by Design</u>	
<u>2.5 Secure Implementation</u>	
<u>2.6 Use of Dependencies</u>	
<u>2.7 Tests</u>	
<u>2.8 Project Documentation</u>	
<u>3.0 Key Findings Table</u>	8
<u>4.0 Findings</u>	10
<u>4.1 Duplicate Liquidations Can Drain Protocol Assets</u>	
Critical	Fixed
<u>4.2 Risk of Over-Borrowing Beyond Deposits from Decimal Precision Limits</u>	
High	Fixed
<u>4.3 Failure to Update Interest Prevents Liquidation in execute_liquidate_initial Function</u>	
High	Fixed
<u>4.4 Free Borrowing by Borrowing at the Start and Repaying at the End of a Time Chunk</u>	
High	Fixed
<u>4.5 Collateral Can Be Lost if Added During the Liquidation Process</u>	
High	Fixed
<u>4.6 Inconsistent Yield Accumulation Across Multiple Supplies</u>	
High	Fixed
<u>4.7 Borrow Duration Expiry Can Trigger Early Liquidation</u>	
High	Fixed
<u>4.8 Withdrawal Request Failure from Token Contract State Changes</u>	
Medium	Fixed
<u>4.9 pow Function Panics on MAXIMUM_POSITIVE_EXPONENT</u>	
Medium	Fixed
<u>4.10 Expired Borrow Position Can Be Repaid Partially</u>	
Medium	Fixed
<u>4.11 Collateral Can Be Added to an Expired Borrow Position</u>	
Medium	Fixed
<u>4.12 Supply Position Spamming Can Disrupt Withdrawal Queue Processing</u>	
Medium	Fixed
<u>4.13 Maximum Borrow Limit Check Not Properly Enforced</u>	
Medium	Fixed
<u>4.14 Reliance on a Single Oracle Source</u>	
Medium	Not Fixed



4.15 Insufficient Safety Margin Between Borrowing and Liquidation Thresholds

✓ Low

✦ Partially Fixed

4.16 Inactive Positions Cause Excessive Snapshot Iteration and Gas Usage

✓ Low

✓ Fixed

4.17 Collateral Asset Can be Added as Borrow Asset

⇓ None

✓ Fixed

4.18 Missing Check

⇓ None

✦ Partially Fixed

4.19 Redundant Check

⇓ None

✓ Fixed

5.0 Appendix A	22
----------------	----

5.1 Severity Rating Definitions

6.0 Appendix B	23
----------------	----

6.1 Thesis Defense Disclaimer

# About Thesis Defense

Defense is the security auditing arm of Thesis, Inc., the venture studio behind tBTC, Fold, Mezo, Acre, Tahoe, Etcher, and Embody. At Defense, we fight for the integrity and empowerment of the individual by strengthening the security of emerging technologies to promote a decentralized future and user freedom. Defense is the leading Bitcoin applied cryptography and security auditing firm. Our team of security auditors have carried out hundreds of security audits for decentralized systems across a number of ecosystems including Bitcoin, Ethereum + EVMs, Stacks, Cosmos SDK, NEAR and more. We offer our services within a variety of technologies including smart contracts, bridges, cryptography, node implementations, wallets and browser extensions, and dApps.

Defense will employ the Defense Audit Approach and Audit Process to the in scope service. In the event that certain processes and methodologies are not applicable to the in scope services, we will indicate as such in individual audit or design review SOWs. In addition, Thesis Defense provides clear guidance on successful Security Audit Preparation.

## Section 1.0 Scope

### Technical Scope

- **Repository:** <https://github.com/Templar-Protocol/contract-mvp>
- **Audit Commit:** `aeb8a93d8844b6b9bb17187e6d9172b713c611fd`
- **Verification Commit:** `4cdab4e75344df814d30715456c5970d686a1698`

### Documentation

- README.md: <https://github.com/Templar-Protocol/contract-mvp/blob/dev/README.md>



# Section 2.0

## Executive Summary

### Schedule

This security audit was conducted from April 2, 2025 to April 29, 2025 by 2 senior security auditors for a total of 8 person weeks.

### Overview

The Templar Protocol is a decentralized, modular borrowing platform developed on the NEAR Protocol, enabling users to facilitate over-collateralized loans. It offers customizable borrowing options, including adjustable interest rates, collateralization ratios, origination fees, and an integrated insurance fund. Key components of the protocol's operations encompass the borrowing process, a liquidation framework, and yield generation for liquidity suppliers. These elements collectively support the protocol's solvency, promote aligned incentives among participants, and maintain overall system stability.

### Threat Model

This threat model outlines the areas investigated during the security assessment of the Templar NEAR Protocol smart contracts. The objective was to evaluate the protocol's resilience against financial, technical, and operational threats, with a focus on ensuring secure-by-design principles and sound implementation. We investigated several areas including:

### Borrowing and Collateralization Logic

- Enforcement of minimum collateral ratios across borrow actions.
- The precision and reliability of price-based collateral and borrow value calculations.
- Interest accrual mechanisms and their responsiveness to timing of borrow and repayment events.
- Behavior of borrow positions nearing or reaching their expiration.
- The protocol's handling of partial repayments and additional collateral on expired positions.
- The ability to adjust collateral after borrowing and implications for liquidation thresholds.

### Liquidation Processes and Synchronization

- The lifecycle of a liquidation event from initiation to finalization.
- Synchronization guarantees around concurrent liquidation attempts.
- Protocol behavior when collateral is added to a position under active liquidation.
- Conditions under which a position becomes eligible or ineligible for liquidation.
- Handling of race conditions between borrower and liquidator actions.

### Yield Accrual and Snapshot System

- Snapshot-based interest and yield tracking for both supply and borrow positions.
- The initialization and updating of snapshot indices across user interactions.
- The effect of timing on interest and yield allocation, including during the current and finalized snapshot periods.
- Gas implications of snapshot iteration over inactive or dormant positions.
- Lifecycle management of supply and borrow positions in storage.

### Withdrawal Queue Design

- The structure and operation of the withdrawal queue, including request sequencing.
- The impact of paused token contracts or disallowed users on queue progression.
- Limitations of the queue in edge cases involving unfulfillable requests.
- Mechanisms for supply withdrawal execution and queue prioritization.
- Susceptibility of the queue to abuse via high-frequency or low-value requests.



## Protocol Configuration and Market Creation

- Validation of asset configurations during market creation.
- Constraints around borrowable and collateralizable asset combinations.
- Enforcement of risk parameters such as maximum borrow limits and collateralization thresholds.
- Behavior of the protocol under custom or misconfigured market parameters.
- Consistency of borrow limits with outstanding user debt.

## Oracle Integration and Price Feed Reliability

- The protocol's use of the Pyth oracle for real-time asset pricing.
- Dependencies on oracle data for borrowing, liquidation, and valuation operations.
- Resilience of the protocol in scenarios involving stale or unavailable price data.
- Assumptions made around oracle update frequency and data integrity.
- The role of oracle confidence intervals and exponent scaling in calculations.

## Code Quality and Logic Validation

- Redundancies and missing validations in critical contract logic.
- Edge cases in configuration parameter handling.
- Consistency of business logic enforcement across contract modules.
- Potential inconsistencies in interest and fee tracking logic.
- Modularity and maintainability of helper and utility functions.

## Security by Design

A number of opportunities have been identified to enhance the protocol's safety and resilience by refining its foundational design choices. Collateral management currently allows users to approach liquidation thresholds with relatively narrow safety margins. Introducing a buffer between borrowing and liquidation limits could help protect users from unnecessary liquidations triggered by minor price fluctuations, while still preserving protocol flexibility. The reliance on a single oracle (Pyth) introduces a potential availability risk during rare service disruptions or network outages. Integrating an additional oracle or introducing fallback mechanisms would further strengthen the reliability of core operations like pricing, liquidation, and borrowing. In certain configurations, it's possible to create markets where borrowing and collateral assets are the same, or where new borrow positions may be immediately eligible for liquidation. These edge cases appear to stem from missing validation checks and can be addressed with lightweight enhancements to the configuration logic. The withdrawal queue design, while efficient in normal scenarios, may face delays if a user is disallowed or if the token contract is paused. Additionally, the low cost of creating supply positions could allow malicious actors to spam the queue. Introducing minimal thresholds for supply and withdrawal actions, along with mechanisms to handle edge cases in queue processing, could improve overall robustness. Allowing partial repayments or added collateral on expired borrow positions introduces scenarios where users may unintentionally lose funds. Aligning behavior in these situations with borrower expectations—for example, requiring full repayment before further actions—would help mitigate these risks.

## Secure Implementation

At the implementation level, the contracts are generally well-structured and follow good practices. Several refinements are suggested to address potential inconsistencies and edge cases. Due to the high precision of the `Decimal` type, certain borrow calculations may inadvertently round down to zero in rare cases, enabling users to bypass collateral ratio checks. This is a subtle but important edge case that can be resolved with additional validation around exponent values. The interest accrual system uses finalized snapshots, which are efficient for performance but may allow a user to borrow and repay within a single chunk without accruing interest. While likely unintended, this behavior could be mitigated by incorporating pro-rated interest for in-chunk borrowing durations, particularly at the point of repayment. In the current liquidation process, it is technically possible for multiple liquidators to simultaneously act on the same position. Adding a locking or status-check mechanism during the liquidation request phase would help prevent overlapping actions and ensure consistent state transitions. Snapshot indexing across supply positions varies slightly between initial and subsequent deposits, which may lead to small yield



imbalances. Aligning this behavior and considering pro-rated yield calculation would ensure a more equitable distribution of rewards across all suppliers. Inactive or closed positions are currently retained in storage, which can lead to increased gas usage when reactivated after long periods. Cleaning up these positions when appropriate—while returning unused storage—would improve efficiency and reduce user friction. Finally, a few functions contain checks that could either be removed for clarity or enhanced to ensure logic matches intended behavior—for instance, properly accounting for already borrowed amounts when applying maximum borrow limits.

## Use of Dependencies

The contract exclusively relies on the Pyth oracle for obtaining the prices of collateral and borrowed assets, which may cause potential risk. Furthermore, it is dependent on the state of the Token contract, potentially affecting the withdrawal process, as detailed in the findings.

## Tests

The implementation has good test coverage but significant gaps remain that could impact the contract's reliability, particularly concerning the scenarios outlined in the findings. Strengthening the test suite is strongly recommended to ensure more robust protection against potential vulnerabilities.































## Project Documentation

No documentation was provided for this project.



## Section 3.0

# Key Findings Table

Issues	Severity	Status
ISSUE #1 Duplicate Liquidations Can Drain Protocol Assets	 Critical	 Fixed
ISSUE #2 Risk of Over-Borrowing Beyond Deposits from Decimal Precision Limits	 High	 Fixed
ISSUE #3 Failure to Update Interest Prevents Liquidation in <code>execute_liquidate_initial</code> Function	 High	 Fixed
ISSUE #4 Free Borrowing by Borrowing at the Start and Repaying at the End of a Time Chunk	 High	 Fixed
ISSUE #5 Collateral Can Be Lost if Added During the Liquidation Process	 High	 Fixed
ISSUE #6 Inconsistent Yield Accumulation Across Multiple Supplies	 High	 Fixed
ISSUE #7 Borrow Duration Expiry Can Trigger Early Liquidation	 High	 Fixed
ISSUE #8 Withdrawal Request Failure from Token Contract State Changes	 Medium	 Fixed
ISSUE #9 <code>pow</code> Function Panics on <code>MAXIMUM_POSITIVE_EXPONENT</code>	 Medium	 Fixed
ISSUE #10 Expired Borrow Position Can Be Repaid Partially	 Medium	 Fixed
ISSUE #11 Collateral Can Be Added to an Expired Borrow Position	 Medium	 Fixed
ISSUE #12 Supply Position Spamming Can Disrupt Withdrawal Queue Processing	 Medium	 Fixed
ISSUE #13 Maximum Borrow Limit Check Not Properly Enforced	 Medium	 Fixed
ISSUE #14 Reliance on a Single Oracle Source	 Medium	 Not Fixed
ISSUE #15 Insufficient Safety Margin Between Borrowing and Liquidation Thresholds	 Low	 Partially Fixed
ISSUE #16 Inactive Positions Cause Excessive Snapshot Iteration and Gas Usage	 Low	 Fixed





Issues	Severity	Status
ISSUE #17 Collateral Asset Can be Added as Borrow Asset	None	Fixed
ISSUE #18 Missing Check	None	Partially Fixed
ISSUE #19 Redundant Check	None	Fixed

Severity definitions can be found in [Appendix A](#)

# Section 4.0

## Findings

We describe the security issues identified during the security audit, along with their potential impact. We also note areas for improvement and optimizations in accordance with best practices. This includes recommendations to mitigate or remediate the issues we identify, in addition to their status before and after the fix verification.

ISSUE#1

### Duplicate Liquidations Can Drain Protocol Assets

⬆ Critical

✅ Fixed

#### Location

[contract/market/src/impl\\_helper.rs#L320](#)

[common/src/borrow.rs#L78-L84](#)

[common/src/borrow.rs#L518-L522](#)

#### Description

The smart contract does not prevent multiple liquidators from initiating liquidation on the same borrow position simultaneously. As a result, multiple liquidation requests can proceed in parallel and all may be finalized, even after the position has already been fully liquidated. The smart contract doesn't check if multiple liquidators are liquidating a borrow position simultaneously resulting in all liquidation requests being processed.

#### Impact

Once the first liquidation request is processed, the borrow position is fully reset via the `full_liquidation` function:

```
pub(crate) fn full_liquidation(&mut self, current_snapshot_index: u32) {
    self.liquidation_lock = false;
    self.started_at_block_timestamp_ms = None;
    self.collateral_asset_deposit = 0.into();
    self.borrow_asset_principal = 0.into();
    self.borrow_asset_fees.clear(current_snapshot_index);
}
```

However, other liquidation requests continue through finalization. Since the position's `borrow_asset_principal` has already been reset to zero, any additional `borrow_asset_amount` sent by other liquidators is treated as recovered yield and distributed across protocol accounts and supply positions — even though the collateral was already transferred to the first liquidator. Additionally, this violates the protocol's collateral asset invariant by effectively distributing the same collateral multiple times. This leads to collateral imbalances, which over time could make it impossible to repay or liquidate other positions due to depleted collateral reserves.

#### Recommendation

We recommend introducing a check during the liquidation request phase to ensure that the position is not already in the process of being liquidated. This will prevent duplicate liquidation attempts and ensure consistent protocol state and collateral accounting.



ISSUE#2

## Risk of Over-Borrowing Beyond Deposits from Decimal Precision Limits

High

Fixed

### Location

[common/src/market/configuration.rs#L176](#)

### Description

When a user borrows against deposited collateral, the smart contract enforces the minimum collateral ratio requirement by validating the following condition: `scaled_collateral_value >= scaled_borrow_value * MCR`. The `scaled_borrow_value` is computed as: `scaled_borrow_value = borrow_amount * ((self.price - self.confidence) * self.power_of_10)`, where `power_of_10` is calculated as: `power_of_10 = Decimal::TEN.pow(pyth_price.expo - decimals)`. Due to the Decimal type's 38-digit precision limit, if `(pyth_price.expo - decimals)` results in less than -38, the computed `power_of_10` becomes 0. This leads to the entire `scaled_borrow_value` being evaluated as 0, irrespective of the actual `borrow_amount`. As a result, the borrower's scaled collateral value will always satisfy the MCR check.

### Impact

Users can over borrow beyond their collateral limits, resulting in draining of the pool's liquidity.

### Recommendation

We recommend validating an asset's decimal value and exponent to ensure that their difference does not result in a value less than -38 when calculating `power_of_10`, prior to creating a market for the asset.

ISSUE#3

## Failure to Update Interest Prevents Liquidation in `execute_liquidate_initial` Function

High

Fixed

### Location

[contract/market/src/impl\\_helper.rs#L71](#)

### Description

In the `execute_liquidate_initial()` function, the latest accumulated interest is not factored in because `accumulate_interest()` is not called on the borrow position. This oversight leaves the borrow position outdated, so when `borrow_position.can_be_liquidated(price_pair, env::block_timestamp_ms())` is invoked, it may return `false`, even if the position should be eligible for liquidation.

### Impact

If the position isn't liquidated in time, the platform may face increased risk. In the case of a significant market downturn, the borrower's collateral might continue to lose value, making it even harder for the platform to recover the loaned amount, leading to bad debt.

### Recommendation



We recommend calling `accumulate_interest()` on the borrow position before checking for the liquidation.

ISSUE#4

## Free Borrowing by Borrowing at the Start and Repaying at the End of a Time Chunk

High

Fixed

### Location

[common/src/borrow.rs#L72](#)

[common/src/market/impl.rs#L205](#)

[common/src/market/impl.rs#L142](#)

[common/src/borrow.rs#L262](#)

[common/src/borrow.rs#L244](#)

### Description

The borrowing system relies on `finalized_snapshots` to calculate interest on positions. These snapshots reflect accrued interest in previous time chunks only. The `current_snapshot` (from the ongoing time chunk) is excluded from this calculation. This allows a borrower to:

1. Borrow at the start of a time chunk.
2. Repay just before the chunk ends.
3. Pay zero interest, since no finalized snapshot includes their borrow duration.

Currently, each borrow position sets its `next_snapshot_index` to `finalized_snapshots.len()`, which corresponds to the `current_snapshot_index`, both when the position is created and when interest is being calculated (assuming `max_snapshot_limit`). During interest calculation, this index is skipped causing interest from the current time chunk to be completely ignored.

### Impact

Borrowers can exploit this loophole to borrow without incurring any fees, leading to potential financial loss or system imbalance.

### Recommendation

For a short-term fix we recommend setting `next_snapshot_index` to `finalized_snapshots.len() - 1`. This will charge interest starting from one snapshot earlier, even if the borrower wasn't active at that time. For an accurate long-term fix we recommend creating a separate function to:

- Calculate pro-rated interest for the borrow duration within the current time chunk.
- Use this only at the time of repayment.
- Combine this with interest from finalized snapshots. This approach ensures borrowers pay fair interest without compromising the current snapshot-based system. However, this solution must be handled carefully when dealing with partial repayments.



ISSUE#5

## Collateral Can Be Lost if Added During the Liquidation Process

High

Fixed

### Location

[contract/market/src/impl\\_ft\\_receiver.rs#L50](#)

[contract/market/src/impl\\_helper.rs#L320](#)

[contract/market/src/impl\\_helper.rs#L323](#)

[common/src/borrow.rs#L78-L84](#)

### Description

During the collateral deposit process, if the `execute_collateralize` function is called immediately after the collateral has been liquidated but before the `liquidate_ft_transfer_call_02_finalize` function is executed in the liquidation process (e.g., when the `execute_collateralize` transaction occurs in the same block after the collateral transfer), the deposited collateral will be lost. This happens because the `liquidate_ft_transfer_call_02_finalize` function performs a full liquidation, clearing all elements of the borrow position except for the `temporary_lock`.

### Impact

Borrowers may attempt to race liquidators by quickly adding collateral to avoid liquidation. However, in doing so, they risk losing any collateral added during this race, on top of the previous collateral, if the liquidation is finalized first.

### Recommendation

We recommend preventing the addition of collateral to a borrow position while it is being liquidated to avoid the loss of newly added collateral.

ISSUE#6

## Inconsistent Yield Accumulation Across Multiple Supplies

High

Fixed

### Location

[common/src/market/impl.rs#L142](#)

[common/src/supply.rs#L34](#)

[common/src/supply.rs#L136](#)

[common/src/supply.rs#L129](#)

### Description

According to the specification, yield accumulation for a supply position should begin from the next snapshot, since the supplier may not have been active for the entire duration of the current snapshot. This logic is correctly enforced during the initial creation of a supply position by setting `next_snapshot_index` to `finalized_snapshots.len() + 1 (current_snapshot + 1)`. However, for subsequent deposits into an existing supply position in later snapshots, this behavior is not maintained. Instead, `next_snapshot_index` is calculated as just `finalized_snapshots.len()`



(assuming max `snapshot_limit` ). As a result, yield starts accumulating from the current snapshot, even though the supply was not present for its full duration. This behavior is inconsistent and leads to unfair yield allocation: the supplier receives full snapshot yield in some cases and is deprived of proportional yield in others.

## Impact

On subsequent supplies made in later snapshots, the supplier incorrectly receives yield for the entire current snapshot, even if the supply was added partway through. Conversely, during the initial supply, the supplier is deprived of yield for the time their funds were active within the current snapshot. In a rare edge case, a malicious actor could exploit this behavior by repeatedly supplying near the end of a snapshot and withdrawing at the start of the next. This cycle could allow them to accumulate yield without meaningful participation in the snapshot duration.

## Recommendation

To ensure fair yield distribution, we recommend calculating the yield for the current snapshot proportionally, based on the duration each individual supply has been active within that snapshot. This prevents over-rewarding later supplies and ensures that early supplies are not unfairly deprived of yield.

## Verification Status

Initial review of the changes over several hours indicates that the issue has been resolved. However, due to significant design changes, and the addition of substantial lines of new code to resolve the issue, a comprehensive verification of all of the changes could not be completed within the available timeframe.

We recommend that these changes be audited, and estimate that it would require an additional 2-3 days for a single security auditor.

ISSUE#7

## Borrow Duration Expiry Can Trigger Early Liquidation

⬆ High

✓ Fixed

## Description

When the `borrow_maximum_duration_ms` is configured to limit the maximum duration of a borrow, a user may repay nearly the full loan, leaving a negligible 'dust' amount below the `minimum_borrow_amount` , and withdraw the maximum collateral allowed. If the configured maximum borrow duration expires while the dust amount remains, and the user subsequently adds collateral and attempts to borrow again, they become eligible for immediate liquidation due to the expired duration on the previous borrow. This issue can also arise in standard borrowing situations where an account borrows multiple times, as subsequent borrowings do not extend the position's expiry, leading to potential early liquidations.

## Impact

The borrow position may be liquidated immediately or earlier than expected, even if the user's current borrow is still active.

## Recommendation

We recommend implementing a strategy to address this issue by either preventing additional borrowing until full repayment of the previous loan is made or by extending the expiry duration accordingly for subsequent borrowings.



ISSUE#8

## Withdrawal Request Failure from Token Contract State Changes

Medium

Fixed

### Location

[contract/market/src/impl\\_market\\_external.rs#L188-L193](#)

### Description

Asset suppliers must complete a two-step process to withdraw their funds:

- Submit a withdrawal request using `create_supply_withdrawal_request`.
- Execute the withdrawal using `execute_next_supply_withdrawal_request`.

When a withdrawal request is created, it is appended to a queue structured as a doubly linked list. Withdrawal requests are processed sequentially — the request at the front of the queue must be fulfilled before subsequent requests can be executed.

If, after submitting a withdrawal request, the user is placed on a disallow list by the token contract or the token contract is paused, the `execute_next_supply_withdrawal_request` call will fail when attempting to process that user's request.

### Impact

If the token contract is paused, it is a temporary state that should revert to normal relatively quickly, causing only a short-term impact on the queue. However, if a user is disallowed, this status persists for an extended period, effectively rendering the queue unusable.

### Recommendation

Implement functionality to detect and handle disallowed users by skipping their withdrawal request and continuing with the processing of the queue. The protocol can then decide how to address the disallowed request. This request can either be retried later or processed in another manner, depending on the protocol's requirements.

ISSUE#9

## pow Function Panics on MAXIMUM\_POSITIVE\_EXPONENT

Medium

Fixed

### Location

[common/src/number.rs#L136](#)

### Description

The maximum allowed exponent value is 95, but when this value is used with `Decimal::TEN.pow()`, the `pow()` function panics. This occurs because the maximum number of fully-representable whole digits for a 384-bit number is 115, yet for decimals, it is calculated as 20 instead of 38, leading to an arithmetic operation overflow.

### Impact

During the computation of `power_of_10` using `Decimal::TEN.pow(pyth_price.expo - decimals)`, if the evaluated expression `pyth_price.expo - decimals` equals 95, the operation will panic. This behavior



can block critical user actions such as borrowing, liquidation, and collateral withdrawal. In particular, users might be unable to withdraw collateral amounts when its value gets increased while still maintaining their borrow position.

## Recommendation

We recommend modifying the `MAXIMUM_POSITIVE_EXPONENT` value to prevent the arithmetic operation overflow.

ISSUE#10

## Expired Borrow Position Can Be Repaid Partially

Medium

Fixed

## Description

The smart contract allows for partial or full repayment of expired borrow positions. However, if a position is only partially repaid, it may be liquidated shortly thereafter—before the full repayment is completed. This could result in the borrower losing both the partial repayment and the collateral.

## Impact

Partial repayment of expired borrow positions can lead to the loss of the partial repayment if the position is liquidated before the full repayment is made.

## Recommendation

To prevent the loss of partial repayments, we recommend that expired borrow positions be repaid in full only.

## Verification Status

Fixed.

ISSUE#11

## Collateral Can Be Added to an Expired Borrow Position

Medium

Fixed

## Description

The smart contract does not prevent the addition of collateral when the borrow position is in the `BorrowStatus::Liquidation(LiquidationReason::Expiration)` state. This added collateral can be liquidated alongside the original collateral.

## Impact

Allowing collateral to be added to an expired borrow position can result in the loss of the additional collateral if the position is not fully repaid before liquidation.

## Recommendation

We recommend preventing the addition of collateral to expired borrow positions to prevent the loss of additional collateral during liquidation.





ISSUE#12

## Supply Position Spamming Can Disrupt Withdrawal Queue Processing

Medium

Fixed

### Description

Creating supply positions is relatively inexpensive, which allows malicious actors to create a large number of supply positions. This could flood the withdrawal queue with an overwhelming number of withdrawal requests, congesting the queue and preventing legitimate withdrawal requests from being processed in a timely manner.

### Impact

The withdrawal queue can become congested with excessive requests, leading to delays in processing legitimate withdrawal requests. This disrupts the proper functioning of the withdrawal system, making it difficult for users to access their funds in a timely manner.

### Recommendation

To mitigate this issue, we recommend implementing a minimum supply amount and a minimum withdrawal amount to prevent the queue from being flooded by low-value supply positions, ensuring that legitimate withdrawal requests are processed efficiently.

ISSUE#13

## Maximum Borrow Limit Check Not Properly Enforced

Medium

Fixed

### Location

[contract/market/src/impl\\_market\\_external.rs#L80-L83](#)

### Description

When a user borrows, there is a check in place to enforce the maximum borrowable amount. However, this check does not account for the amount the user has already borrowed, potentially allowing them to exceed the intended borrowing limit.

### Impact

This oversight could allow users to borrow more than the maximum allowed amount, potentially disrupting the lending system's balance and exposing it to greater risk.

### Recommendation

We recommend updating the maximum borrowable amount check to include the user's current outstanding borrow amount, ensuring the limit is enforced accurately.



ISSUE#14

## Reliance on a Single Oracle Source

Medium

Not Fixed

### Location

[common/src/market/balance\\_oracle\\_configuration.rs#L26](#)

### Description

The contract depends on the Pyth oracle to obtain price data for both borrowed and collateral assets. This introduces a single point of failure risk: if the oracle service becomes unavailable due to network disruptions, blockchain issues, or problems with underlying data sources, the protocol may be unable to access updated or accurate pricing information.

From Pyth documentation:

A network outage (at the internet level, blockchain level, or at multiple data providers) may prevent the protocol from producing new price updates. (Such outages are unlikely, but integrators should still be prepared for the possibility.) In such cases, Pyth may return a stale price for the product.

### Impact

It will impact critical operations like collateral valuation, liquidation processes, and overall system stability.

### Recommendation

To reduce dependency risk, it is recommended to integrate a secondary oracle or implement fallback mechanisms to ensure continued access to reliable price data during outages or failures.

### Verification Status

The Templar team has confirmed that their smart contract currently supports only a single oracle provider and does not support integration with additional oracles at this time.

ISSUE#15

## Insufficient Safety Margin Between Borrowing and Liquidation Thresholds

Low

Partially Fixed

### Location

[contract/market/src/impl\\_helper.rs#L372-L375](#)

### Description

The current collateral management system enforces two checks: **Initial Borrowing Check**: Ensures users cannot borrow beyond a predefined collateral ratio. **Collateral Withdrawal Check**: Allows users to withdraw collateral until their position nears the liquidation threshold, exposing them to significant liquidation risk. While users choose this risk, the protocol allows excessively tight margins, encouraging avoidable liquidations. Although liquidations generate revenue, the protocol should prioritize protecting users from unnecessary risks. The current collateral management mechanism enforces two distinct checks: **Initial Borrowing Check**: Ensures users cannot borrow beyond a predefined minimum collateral ratio (e.g., maximum LTV). **Collateral Withdrawal Check**: Allows users to withdraw collateral until their position



approaches the liquidation threshold, significantly increasing liquidation risk. This design permits users to:

- Borrow up to the initial collateral limit, then
- Withdraw additional collateral until their position is dangerously close to liquidation. While users voluntarily assume this risk, the protocol incentivizes avoidable liquidations by permitting excessively tight margins. Although liquidations generate revenue, the protocol should prioritize user protection by discouraging unnecessarily risky positions.

## Impact

- **High Liquidation Risk:** Users can withdraw collateral close to the liquidation threshold, risking liquidation from minor market fluctuations.
- **Poor User Experience:** Borrowers may unintentionally over-leverage due to insufficient safeguards.
- **Protocol Reputation Risk:** Excessive liquidations could erode trust in the system's stability.

## Recommendation

To mitigate unnecessary liquidations while maintaining flexibility, we recommend introducing a safety buffer between the initial collateral ratio and the liquidation threshold. For example, if the liquidation threshold is 80% LTV, restrict withdrawals if the position exceeds 70% LTV, ensuring a 10% buffer. This balances user autonomy with effective risk mitigation.

## Verification Status

An additional check now enforces that both the initial and minimum collateral ratios are met during withdrawals. However, the recommended safety buffer to reduce liquidation risk resulting from the equality of MCR and IMR has not been implemented.

ISSUE#16

## Inactive Positions Cause Excessive Snapshot Iteration and Gas Usage

✓ Low

☑ Fixed

## Location

[common/src/supply.rs#L115](#)

[common/src/borrow.rs#L222](#)

[common/src/supply.rs#L123-L137](#)

[common/src/borrow.rs#L239-L263](#)

## Description

When supply or borrow positions are closed but not removed from storage, returning users may face inefficiencies. If a user reactivates a previously closed position after a long period, the `next_snapshot_index` could be outdated, requiring the contract to iterate over a large number of snapshots to catch up to the `current_snapshot_index`. This can lead to out-of-gas errors and high unnecessary gas costs. While this issue can be mitigated by manually calling `harvest_yield` for supply positions and `apply_interest` for borrow positions in smaller steps, this is tedious and costly for users. Additionally, since the positions are effectively inactive (e.g., `borrow_asset_deposit` and `principal` are zero), keeping them in storage serves little purpose.



## Impact

- Users may encounter out-of-gas errors when reactivating old positions.
- Users incur unnecessary gas costs for large snapshot iterations.
- User experience is degraded due to the manual steps required to recover positions.
- Storage is inefficiently used, since closed positions are retained unnecessarily.

## Recommendation

We recommend removing supply and borrow positions from storage when they are closed or liquidated, and returning the unused storage to the user. Additionally, we recommend skipping snapshot iteration for inactive positions ( `borrow_asset_deposit == 0` for supply, `principal == 0` for borrow), and setting `next_snapshot_index` based on the current `finalized_snapshots.len()` .

ISSUE#17

## Collateral Asset Can be Added as Borrow Asset

None

Fixed

## Location

[common/src/market/configuration.rs#L86](#)

## Description

When creating a new market, `configuration.validate()` does not check if the borrowing asset and collateral asset are the same, which could result in a pool being created where the borrowing and collateral asset are identical.

## Impact

This behavior is not intended to be supported by contract.

## Recommendation

We suggest incorporating a check within `configuration.validate()` to prevent using the same asset for both collateral and borrowing.

ISSUE#18

## Missing Check

None

Partially Fixed

## Location

[common/src/market/configuration.rs#L86](#)

## Description

The `validate` function in `configuration.rs` is missing a check to ensure that `borrow_mcr_initial` (the minimum collateral ratio required at the time of borrowing) is greater than `borrow_mcr` (the minimum collateral ratio used for ongoing maintenance and liquidation). Without this validation, configurations could be set where newly created borrow positions are immediately eligible for liquidation.



## Impact

Improper configuration could result in borrow positions being instantly liquidatable upon creation, undermining protocol safety and user trust.

## Recommendation

Add a validation check to ensure that `borrow_mcr_initial` is strictly greater than `borrow_mcr` to maintain proper collateralization logic and to prevent unintended liquidations.

## Verification Status

A validation check has been added to ensure that `borrow_mcr_initial` is not less than `borrow_mcr`. However, this only enforces a non-strict inequality (  $\geq$  ) rather than the recommended strict inequality (  $>$  ). As a result, the fix does not fully align with the intended safeguard against edge cases that could compromise collateralization guarantees.

ISSUE#19

## Redundant Check

None

Fixed

## Location

[common/src/market/impl.rs#L273](#)

## Description

The referenced check is redundant and does not contribute to any functional logic or safety guarantees.

## Impact

No impact.

## Recommendation

We recommend removing the redundant check to simplify the codebase and improve readability.








# Section 5.0

## Appendix A

### Severity Rating Definitions

At Thesis Defense, we utilize the [Immunefi Vulnerability Severity Classification System - v2.3](#).

Severity	Definition
 Critical	<ul style="list-style-type: none"><li>• Manipulation of governance voting result deviating from voted outcome and resulting in a direct change from intended effect of original results</li><li>• Direct theft of any user funds, whether at-rest or in-motion, other than unclaimed yield</li><li>• Direct theft of any user NFTs, whether at-rest or in-motion, other than unclaimed royalties</li><li>• Permanent freezing of funds</li><li>• Permanent freezing of NFTs</li><li>• Unauthorized minting of NFTs</li><li>• Predictable or manipulable RNG that results in abuse of the principal or NFT</li><li>• Unintended alteration of what the NFT represents (e.g. token URI, payload, artistic content)</li><li>• Protocol insolvency</li></ul>
 High	<ul style="list-style-type: none"><li>• Theft of unclaimed yield</li><li>• Theft of unclaimed royalties</li><li>• Permanent freezing of unclaimed yield</li><li>• Permanent freezing of unclaimed royalties</li><li>• Temporary freezing of funds</li><li>• Temporary freezing NFTs</li></ul>
 Medium	<ul style="list-style-type: none"><li>• Smart contract unable to operate due to lack of token funds</li><li>• Enabling/disabling notifications</li><li>• Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol)</li><li>• Theft of gas</li><li>• Unbounded gas consumption</li></ul>
 Low	<ul style="list-style-type: none"><li>• Contract fails to deliver promised returns, but doesn't lose value</li></ul>
 None	<ul style="list-style-type: none"><li>• We make note of issues of no severity that reflect best practice recommendations or opportunities for optimization, including, but not limited to, gas optimization, the divergence from standard coding practices, code readability issues, the incorrect use of dependencies, insufficient test coverage, or the absence of documentation or code comments.</li></ul>



# Section 6.0

## Appendix B

### Thesis Defense Disclaimer

Thesis Defense conducts its security audits and other services provided based on agreed-upon and specific scopes of work (SOWs) with our Customers. The analysis provided in our reports is based solely on the information available and the state of the systems at the time of review. While Thesis Defense strives to provide thorough and accurate analysis, our reports do not constitute a guarantee of the project's security and should not be interpreted as assurances of error-free or risk-free project operations. It is imperative to acknowledge that all technological evaluations are inherently subject to risks and uncertainties due to the emergent nature of cryptographic technologies.

Our reports are not intended to be utilized as financial, investment, legal, tax, or regulatory advice, nor should they be perceived as an endorsement of any particular technology or project. No third party should rely on these reports for the purpose of making investment decisions or consider them as a guarantee of project security.

Links to external websites and references to third-party information within our reports are provided solely for the user's convenience. Thesis Defense does not control, endorse, or assume responsibility for the content or privacy practices of any linked external sites. Users should exercise caution and independently verify any information obtained from third-party sources.

The contents of our reports, including methodologies, data analysis, and conclusions, are the proprietary intellectual property of Thesis Defense and are provided exclusively for the specified use of our Customers. Unauthorized disclosure, reproduction, or distribution of this material is strictly prohibited unless explicitly authorized by Thesis Defense. Thesis Defense does not assume any obligation to update the information contained within our reports post-publication, nor do we owe a duty to any third party by virtue of making these analyses available.

