# Clearpool
## Security Review


Cantina Managed review by:

**High Byte**, Security Researcher
**Slowfi**, Security Researcher



September 4, 2025

# Contents

# 1 Introduction

## 1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

## 1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

## 1.3 Risk assessment

| Severity | Description |
|---|---|
| **Critical** | *Must* fix as soon as possible (if already deployed). |
| **High** | Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users. |
| **Medium** | Global losses <10% or losses to only a subset of users, but still unacceptable. |
| **Low** | Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies. |
| **Gas Optimization** | Suggestions around gas saving practices. |
| **Informational** | Suggestions around best practices or readability. |

### 1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

# 2 Security Review Summary

Clearpool is the leading DeFi credit marketplace pioneering Real-World Asset lending.

From Aug 5th to Aug 7th the Cantina team conducted a review of clearpool-payfi-vaults on commit hash 3f87019c. The team identified a total of **13** issues:

**Issues Found**

| Severity | Count | Fixed | Acknowledged |
|---|---|---|---|
| Critical Risk | 0 | 0 | 0 |
| High Risk | 4 | 4 | 0 |
| Medium Risk | 2 | 2 | 0 |
| Low Risk | 3 | 2 | 1 |
| Gas Optimizations | 0 | 0 | 0 |
| Informational | 4 | 2 | 2 |
| **Total** | **13** | **10** | **3** |

# 3 Findings

## 3.1 High Risk

### 3.1.1 Decimal truncation on asset vaults different to 18 decimals loss precission

**Severity:** High Risk

**Context:** *(No context files were provided by the reviewer)*

**Description:** ***[Found by Clearpool team]***

When using asset of less than 18 decimals, the protocol truncates the exchange rate reducing the shares calculation. This happens on `TellerWithMultiAssetSupport.sol`, `AccountantWithRateProviders.sol` and `AtomicQueue.sol`.

The decimal truncation worsen the accountant of profits with time due to interest compounding.

**Recommendation:** Align decimal calculation and exchange rates across all the shares and assets calculations to minimize precision loss.

**Clearpool:** Fixed in commit 3f87019c. This fix includes new functions to allow vaults to operate with base asset less than 18 decimals and/or exchange asset with different decimals. Rate providers are fixed to return value on 18 decimals. The protocol now operates by escalating all amounts to 18 decimal asset, doing any required calculation and then returned to the original decimal base asset amount. This reduces the decimal loss precision calculation.

**Cantina Managed:** Fix verified.

### 3.1.2 Rate provider return amount inconsistency

**Severity:** High Risk

**Context:** *(No context files were provided by the reviewer)*

**Description:** In commit 3ce9f7a82, the newly introduced functions `calculateSharesForAmount` and `calculateAmountForShares` expected the `rateProvider` to return the value of `getRate` function on 18 decimals. However the `claimFees` functions expected the value to be returned on quote decimals. This inconsistency lead to improper amount calculations on with non pegged assets.

**Recommendation:** Consider using a standard way across the code on how `rateProvider` returns the quoted scaled amounts.

**Clearpool:** Fixed in commit 3f87019c.

**Cantina Managed:** Fix verified. The fix expected all `rateProvider` to return its value scaled to 18 decimals.

### 3.1.3 Incorrect scaling in `_calculateWantAmount`

**Severity:** High Risk

**Context:** *(No context files were provided by the reviewer)*

**Description:** The `_calculateWantAmount` from `AtomicQueue.sol` has three different paths to calculate the amount wanted for the exchange.

The first path may fail to obtain the right amount as the provided amount may be different than 18 decimals. This when multiplying by the rate, scaled to 18 and dividing by `1e18` the resulting amount is scaled to the original asset decimal and not to 18 as expected.

The second path may fail if the vault asset uses 6 decimal asset as it presumes the shares are in 18 decimals and vault asset may use a different scale.

**Recommendation:** For the first path consider using, `ONE_SHARE` for division to ensure returning in 18 decimals.

```
offerAmount.mulDivDown(rate, ONE_SHARE);
```

For the second path consider multiplying by `ONE_SHARE` instead of `1e18` to obtain the shares escaled to vault decimals.

**Clearpool:** Fixed in commit 3f87019c.

**Cantina Managed:** Fix verified.

#### 3.1.4 Inconsistent scaling in `claimFees` may zero out transfers

**Severity:** High Risk

**Context:** *(No context files were provided by the reviewer)*

**Description:** In the non-pegged branch of `claimFees`, the conversion mixes units: fees are scaled to the fee asset's decimals, but division uses a `rate` expected at 18 decimals. With e.g. base = 18 decimals, fee asset = 6 decimals, small accrued fees can round down to zero while `_feesOwedInBase` is still cleared.

**Recommendation:** Consider to align units by replacing with:

```
.mulDivDown(1e18, rate)
```

**Clearpool:** Fixed in commit 3f87019c.

**Cantina Managed:** Fix verified.

### 3.2 Medium Risk

#### 3.2.1 Public `checkpoint()` mutates state while paused

**Severity:** Medium Risk

**Context:** AccountantWithRateProviders.sol#L509-L511

**Description:** When the contract is paused (`state._isPaused = true`), the expected behavior is that sensitive state changes are halted until governance intervention. However, `checkpoint()` remains callable by any address without pause restrictions. Calling `checkpoint()` executes `_checkpointInterestAndFees()`, which updates `_exchangeRate`, `_feesOwedInBase`, and `_lastAccrualTime` based on the current exchange rate. If the pause was triggered by an out-of-bounds exchange rate in `updateExchangeRate`, calling `checkpoint()` during the paused state can still store that invalid rate into contract storage along with associated fee accruals. This undermines the purpose of pausing by committing potentially unsafe values that should remain frozen. For example:

1. Trigger `updateExchangeRate` with an out-of-bounds value so the contract pauses.
2. While paused, call `checkpoint()` from any address.
3. `_exchangeRate` and accrual data are still updated in storage despite the paused state.

This behavior allows invalid or manipulated data to be persisted, potentially affecting redemptions, pricing, or downstream accounting.

**Recommendation:** Consider to prevent `checkpoint()` from executing while the contract is paused, unless called by an explicitly authorized recovery role. This ensures that no invalid exchange rate or accrual data is committed during the paused period, preserving the integrity of the pause mechanism.

**Clearpool:** Added requiresAuth to restrict the call of this function to controlled address. Fixed on commit cb325156.

**Cantina Managed:** Fix verified.

#### 3.2.2 Invalid Exchange Rate Causes Permanent State Corruption in Fee Calculations

**Severity:** Medium Risk

**Context:** AccountantWithRateProviders.sol#L248-L254, AccountantWithRateProviders.sol#L254-L284, AccountantWithRateProviders.sol#L486-L501

**Description:** The `updateExchangeRate()` function stores invalid exchange rates that are subsequently used as the base for all future interest calculations. The function executes `_checkpointInterestAndFees()` before validation, but always stores the new exchange rate.

When an invalid rate is provided:

1. `_checkpointInterestAndFees()` executes with the valid old rate.
2. Validation fails and the contract pauses.
3. The invalid rate is stored.
4. Future calls to `_checkpointInterestAndFees` use this invalid rate as the base.

This means any subsequent checkpoint will calculate interest using potentially invalid data, and the error compounds over time.

**Recommendation:** Consider to enforce pause semantics consistently across all writers:

- Consider to make `_checkpointInterestAndFees()` a no-op when paused and optionally revert `checkpoint()` while paused. This requires to remove the the if control pause flow statement of `updateExchangeRate()` to work.
- Consider to pause-gate any privileged function that indirectly checkpoints or ensure they early-exit without accrual when paused.
- Consider to add tests covering: after a pause, calls to `checkpoint()` or admin setters do not mutate `_exchangeRate`, `_feesOwedInBase`, or `_lastAccrualTime` until unpaused.

**Clearpool:** Fixed on commit cb325156 The fix erase the pause control flow on `updateExchangeRate` to let authorized roles to change the exchange rate in case is improperly set on previous interactions. Having the contract paused prevents further operations from happening, but we still allow to store the checkpoint on subsequent calls as it is consider a potential negligible amount that may not disrupt normal operations due to the off-chain monitoring systems in place.

**Cantina Managed:** Fix verified according to Clearpool team requirements.

## 3.3 Low Risk

### 3.3.1 Insufficient event coverage for state changes

**Severity:** Low Risk

**Context:** AccountantWithRateProviders.sol#L275, AccountantWithRateProviders.sol#L486-L501, TellerWithMultiAssetSupport.sol#L246-L249

**Description:** Several state-changing paths do not emit dedicated events, reducing observability and making off-chain monitoring and accounting harder:

1. Accrual checkpoints: `checkpoint()` calls `_checkpointInterestAndFees()` which mutates `_exchangeRate`, `_feesOwedInBase`, and `_lastAccrualTime` without emitting any event. Accruals triggered indirectly from other functions also leave no checkpoint trace.
2. Share-lock configuration: `setShareLockPeriod(uint64)` updates `shareLockPeriod` without emitting an event, despite affecting transfer restrictions (`beforeTransfer`) and the refundable window used by `refundDeposit`.
3. Pause via `updateExchangeRate`: When bounds/time checks fail, `updateExchangeRate` sets `state._isPaused = true` but does not emit `Paused()` (and duplicates pause logic). This creates inconsistent signaling compared to the dedicated `pause()` function that does emit.

**Recommendation:** Consider to emit explicit events for these transitions and to reuse existing pause logic to avoid divergence. This would align all pause triggers with consistent on-chain signaling, and improve monitoring of accrual and configuration changes.

**Clearpool:** Fixed in commit cb325156.

**Cantina Managed:** Fix verified.

### 3.3.2 Inconsistent enforcement when lowering `maxLendingRate`

**Severity:** Low Risk

**Context:** AccountantWithRateProviders.sol#L320-L323

**Description:** `setMaxLendingRate` only updates the cap and emits `MaxLendingRateUpdated`. If governance **lowers** `maxLendingRate` below the current `lendingInfo._lendingRate`, the contract continues accruing at the now out-of-policy lending rate until a separate `setLendingRate` transaction is sent. This creates a window where accruals don't reflect the newly enforced maximum. Raising the cap is benign.

**Recommendation:** Consider to align the active lending rate with the new cap in the same transaction when the cap is **reduced**:

- Finalize accruals up to the change (i.e., checkpoint first).
- Clamp `lendingInfo._lendingRate` to `min(currentRate, newMax)`.
- Emit a rate-update event to make the adjustment observable.

Alternatively, revert if the new cap is below the current rate unless the call explicitly opts into clamping (e.g., via a parameter), ensuring policy and accrual remain consistent immediately after the cap change.

**Clearpool:** Fixed in commit ed2cb3d9.

**Cantina Managed:** Fix verified.

#### 3.3.3 Solve can be DoSed when a single request is invalid

**Severity:** Low Risk

**Context:** AtomicQueue.sol#L221-L232

**Description/Recommendation:** when a single request is invalid, instead of skipping it the entire solve will revert. It is recommended to `continue` instead. Also noteworthy that `transferFrom` can also fail so it is best to try-catch around that too.

**Clearpool:** Fixed in commit ca64fe22.

**Cantina Managed:** Fix verified.

### 3.4 Informational

#### 3.4.1 Use of raw `1e4` instead of a named constant

**Severity:** Informational

**Context:** AccountantWithRateProviders.sol#L270-L272

**Description:** In `AccountantWithRateProviders`, the checks for `_allowedExchangeRateChangeUpper` and `_allowedExchangeRateChangeLower` use the literal `1e4` as the denominator for basis points calculations. While functionally correct, using a named constant such as `BASIS_POINTS` improves clarity and maintainability, making it explicit that these values are expressed in basis points.

**Recommendation:** Consider to replace the raw `1e4` literal with a descriptive constant (e.g., `uint256 internal constant BASIS_POINTS = 1e4;`) and use it consistently across the codebase for all basis point math. This reduces the risk of misinterpretation and improves code readability.

**Clearpool:** Fixed in commit cb325156.

**Cantina Managed:** Fix verified.

#### 3.4.2 Unused return value and unnecessary computation in `calculateExchangeRateWithInterest`

**Severity:** Informational

**Context:** AccountantWithRateProviders.sol#L383

**Description:** `calculateExchangeRateWithInterest()` returns `(newRate, interestAccrued)`, but across its call sites (`getRate`, `getRateSafe`, `getRateInQuote`, `previewFeesOwed`, and `_checkpointInterestAndFees`) only `newRate` is consumed. This implies the function performs extra work (computing `interestAccrued`) that is not needed for current usage, increasing gas and cognitive load.

**Recommendation:** Consider to decouple responsibilities so callers compute only what they need. For example, expose a lightweight function that returns just the updated rate, and keep a separate path

for callers that require `interestAccrued`. Align call sites to the minimal interface, reducing redundant calculations and improving readability.

**Clearpool:** Acknowledged. Decided to keep this value for off-chain and external integration.

**Cantina Managed:** Acknowledged by Clearpool team.

#### 3.4.3 Increasing Interest Rate creates price drift risk in `AtomicQueue` (mitigated by current withdraw-only usage)

**Severity:** Informational

**Context:** AtomicQueue.sol#L157-L164

**Description:** `AtomicQueue` prices requests at execution time using the current NAV from the accountant. When there is time between request creation and fulfillment, the exchange rate can drift (e.g., due to continuous interest accrual). Example scenario:

1. User creates request: "Buy vault shares with 1,000 USDC".
2. At request time: 1 share = 10 USDC $\rightarrow$ expects ~100 shares.
3. Later, interest accrual raises NAV: 1 share = 10.50 USDC.
4. On execution, user only receives ~95.2 shares instead of 100.

This creates an expectation gap for flows like "buy shares with USDC," where appreciating share price means the user ultimately receives fewer shares than anticipated.

In the current Clearpool protocol, the queue is used **only for withdrawals** (users offer vault shares $\rightarrow$ receive assets), and the **solver role is operated by Clearpool**. In this constrained mode, rate drift tends to work in favor of withdrawing users (more assets per share as time passes) and operational risk is reduced by having the protocol as solver. Nevertheless, the mechanism remains sensitive to time-based price movement and would resurface as a user-facing slippage problem if the queue later supports deposits/buy-share intents.

**Recommendation:**

- Consider to encode user price expectations in requests if bi-directional usage is introduced later:
  - Add a per-request `minAmountOut` (for withdrawals) or `maxSharesPerAsset` / max acceptable rate (for deposits), enforced at fulfillment.
  - Keep short deadlines to limit drift exposure.
- If the intent is withdraw-only, consider to enforce that at the contract level (restrict allowed `offer`/`want` pairs) and document that execution uses real-time NAV so users understand outcomes may improve/change with time.
- For future deposit support, consider UI and solver policies that quote and respect slippage bounds, or snapshot a reference rate at intent time and enforce a tolerance at solve.

**Clearpool:** Acknowledged. This will be a to-do item if Clearpool supports deposit via AtomicQueue in the future. For now it is not an essential fix.

**Cantina Managed:** Acknowledged by Clearpool team.

#### 3.4.4 Bound check incomplete

**Severity:** Informational

**Context:** AccountantWithRateProviders.sol#L205

**Description/Recommendation:** The highlighted line (together with the complementary function) only performs bound check, but does not check if `_allowedExchangeRateChangeUpper > _allowedExchangeRateChangeLower`, basically allowing within-bounds values but in reverse order.

**Clearpool:** Fixed in commit cb325156.

**Cantina Managed:** Fix verified.