Denaria 2

Date	September 2025

1 Executive Summary

This report presents the results of our second engagement with **Denaria-finance** to review their **Decentralized Perpetual Futures Exchange**. Following our initial audit conducted from April 3 to May 20, 2025, this second review was undertaken to assess the current state of the protocol after the team's remediation efforts.

The review was conducted over the course of **three** weeks, from **September 1, 2025** to **September 26, 2025** by **Sergii Kravchenko** and **Arturo Roura**. In the previous assessment, we identified a large number of issues that indicated the codebase was still in its early stages of development. Following that review, the team committed to extensive internal testing and security improvements before returning for this second round of auditing.

The current implementation shows clear progress, with many previous issues addressed and improvements in code structure and protocol design. However, the protocol's highly complex AMM design, individualized accounting, and intricate code structure continue to present significant challenges. The interdependence of subsystems and advanced mathematical operations make it extremely difficult to ensure correctness and security across all scenarios. Testing and validating such a system requires sophisticated invariants and comprehensive monitoring to guarantee that all balances and exposures remain consistent.

Despite the improvements, our review uncovered a number of new major issues at various levels—from impactful typos and logic errors to deeper architectural concerns. The absence of monitoring, circuit breakers, and administrative controls further increases the risk, as there is currently no way to intervene or halt the protocol in the event of an emergency.

While the protocol has evolved, we believe the system in its current form carries a high level of security risk due to its complexity and the challenges inherent in thoroughly testing and monitoring such a design. Without substantial additional work—including adversarial testing, rigorous invariant checks, codebase simplification, and the implementation of robust monitoring and emergency controls—there is a significant risk that severe vulnerabilities could remain undetected and user funds could be at risk if the protocol were deployed to mainnet.

1.1 Post-Audit Fix Review

Following the completion of the audit, the team implemented remediation efforts to address the identified issues. We conducted a review of these fixes in line with the scope and timeframe previously agreed upon, dedicating additional effort to **examine each of the remediations within the available timeframe**. However, several of the fixes were substantial in nature, introducing conceptual changes to key protocol mechanisms. Given the severe time constraints, our review of these remediation efforts **necessarily prioritized breadth over depth**, preventing the thorough analysis that such significant changes warrant. Despite the limited scope of our review, we identified additional issues that emerged as a result of the implemented fixes (see Fix Review Findings). This underscores the ongoing security challenges facing the protocol and reinforces our assessment that the codebase requires further maturation. **We strongly recommend that the team invest additional time in internal security work, comprehensive testing, and engage in another full round of security review or consider a public security competition before any mainnet deployment.**

2 Scope

This review focused on the following repository and code revision:

- **denaria-finance/denaria-perp** 5d143a158d558464e17a4ad9f6d72b83bf42796c

The detailed list of files in scope can be found in the Appendix.

3 Security Considerations

3.1 vAMM Design Complexity and Risk Assessment

The protocol implements multiple sophisticated mechanisms whose interaction creates complexity that expands the attack surface. The vAMM design requires solving third-degree polynomial equations through Newton's method to calculate trade returns while maintaining curve initialization at oracle price. The system must also track liquidity provider positions as they continuously evolve between stable and asset allocations through a matrix-based algorithm that handles flexible LP deposits in arbitrary ratios, including single-sided liquidity provision.

This complexity compounds through the fee distribution mechanism, which must allocate trading fees across LPs with heterogeneous liquidity compositions and temporal positions, creating additional computational and accounting overhead that evolves with each transaction. Since liquidity values can be influenced through trading activity, the curve calculations become susceptible to manipulation through strategic trading patterns, potentially affecting both trade execution and fee distribution. Each additional mechanism - from cubic equation solving to matrix-based LP tracking to dynamic fee distribution - objectively introduces new potential failure points and increases the codebase complexity that must be secured.

In our assessment, the protocol's design choices raise questions about proportionality between complexity and benefits. We observe that the system targets extremely low slippage scenarios near oracle price, which targets a trading regime where the practical outcomes may closely resemble spot price execution. From our perspective, the substantial mathematical sophistication may not provide proportional advantages over simpler approaches, particularly given that the protocol's economic constraints naturally limit viable usage to small-to-medium trades where we believe the practical difference between complex vAMM mechanisms and direct spot price execution could be marginal. We note that the incentive structure discourages large trades relative to pool liquidity, as slippage becomes prohibitive and curve replotting mechanisms incentivize trade fragmentation, further limiting the system to scenarios where we consider simpler mechanisms would likely perform similarly.

This vAMM architecture, with its comprehensive suite of innovative mechanisms, appears to us to represent a high-complexity solution that may introduce disproportionate risks relative to what could potentially be achieved through more straightforward implementations.

▶ **The Denaria team's opinion regarding this security consideration**

3.2 LP Assymetry

The protocol's liquidity provision system fundamentally differs from traditional pooled models by treating LPs as individual passive traders rather than collective liquidity providers. Single-sided deposits and individualized funding fee calculations based on each LP's specific exposure create asymmetries where LPs providing identical amounts can experience vastly different outcomes. Market conditions and subsequent trading activity determine whether an LP earns or pays funding fees, creating race conditions and direct competition between liquidity providers. This design transforms liquidity provision from a straightforward service into a nuanced trading strategy, where LPs must consider market timing, directional exposure, and competitive positioning against other LPs for favorable fee outcomes. The resulting system obscures traditional incentive alignment and creates an environment where liquidity providers compete against each other rather than collectively serving traders.

▶ **The Denaria team's opinion regarding this security consideration**

3.3 Isolated Parallel Accounting

The system relies heavily on parallel accounting, where each user and subsystem maintains its own internal balances. While these balances are expected to remain synchronized, there is little enforcement or reconciliation, resulting in frequent mismatches due to rounding errors. These discrepancies, though small ("dust"), accumulate and are difficult to detect due to the absence of system-wide integrity checks. Some rounding issues were observed that only surface under specific conditions and would be difficult to identify without targeted analysis. The lack of safeguards around accounting mismatches presents a long-term security risk.

▶ **The Denaria team's opinion regarding this security consideration**

3.4 Vault Design

The vault's multi-stablecoin architecture creates operational and stability risks through rigid ratio constraints and assumptions of perpetual dollar parity. The system lacks depeg detection mechanisms, continuing to treat depegged stablecoins as equivalent to $1.00 without emergency response capabilities. This creates arbitrage opportunities, skewed collateral valuations, and systemic risk exposure during stablecoin instability events. Combined with strict ratio enforcement that can block legitimate user operations, the vault design introduces usability friction while remaining vulnerable to collateral devaluation scenarios that could destabilize the entire protocol.

▶ **The Denaria team's opinion regarding this security consideration**

3.5 Absence of Circuit Breakers

The absence of circuit breakers leaves the protocol defenseless against crisis scenarios. Without pause mechanisms or emergency controls, problems compound uncontrollably, bad debt accumulates during liquidation failures, exploits continue draining funds, and market anomalies cascade into protocol-wide instability. Administrators cannot intervene to halt operations or safely transition the system to recovery, forcing the protocol to continue operating in degraded states until external resolution occurs, often resulting in catastrophic damage that proper emergency controls could have prevented.

▶ **The Denaria team's opinion regarding this security consideration**

3.6 Liquidator Exit Risk

The liquidation mechanism creates exit risk for liquidators who must trade their acquired positions back through the curve to realize profits. While liquidators receive discounted positions, they face uncertainty in exit execution due to pool liquidity constraints and market timing. In low-liquidity scenarios, liquidators may be unable to exit positions immediately, exposing them to adverse price movements that can eliminate discount gains. This exit risk may disincentivize liquidation participation precisely when pools have insufficient liquidity, potentially leading to delayed liquidations and bad debt accumulation.

▶ **The Denaria team's opinion regarding this security consideration**

4 Continuous Fuzzing of Existing Invariants

We implemented a continuous fuzzing workflow to identify potential vulnerabilities and ensure the robustness of the Denaria system. The workflow is automated and runs daily on the audited code. Upon completion, a summary report is automatically generated and sent to the development team via Telegram, providing continuous feedback on code stability and invariant violations. This continuous process has been running since September 1, 2025. We did not create new tests/invariants, but relied on existing tests/invariants that were created by the Denaria team.

4.1 Results

- **September 2, 2025:** The fuzzer identified a failing test (`testAddMultiplePrices`), which was triggered by the fuzzer exceeding a bound on the maximum number of rejects for Foundry's `assume` cheat code. The development team promptly provided a fix for the affected test (`testAddMultiplePrices`), and we continued fuzzing on the patched code (commit hash `3187938b8050b1d4e537a7826fc9f07134bd2f4b`).
- **September 10, 2025:** The fuzzer found a second failing test (`testAMMOutputBoundaries`), which was caused by an arithmetic overflow in the curve computations (using Newton's method). The development team reviewed the failing test inputs and concluded that the input values were not in a range that should happen in practice. The fuzzer found a similar failure (i.e., arithmetic overflow in `testAMMOutputBoundaries`) on September 26, 2025.

4.2 Recommendations

We recommend that the Denaria team incorporates the provided fix for the first failure (in `testAddMultiplePrices`) and updates the second failing test (`testAMMOutputBoundaries`) by bounding the input values to a range that is considered "realistic". This ensures that the test accurately reflects the and validates the expected behavior of the AMM for "realistic" conditions. Ideally, what is considered "realistic" should be formalized and arithmetic overflows should be monitored once the system is deployed.

5 Findings

Each issue has an assigned severity:

- Critical issues are directly exploitable security vulnerabilities that need to be fixed.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Issues without a severity are general recommendations or optional improvements. They are not related to security or correctness and may be addressed at the discretion of the maintainer.

5.1 PnL Avoidance Through Zero Exposure Exploit Critical ✓ Fixed

Resolution

Fixed in commit f3869fba8db140ed26fbd92cd39793f0679dc749. In `calcHypotheticalMR` , the order of checks has been changed to return `0` as MR if the position has bad debt, instead of directly assuming that a `0` exposure position is always safe.

Description

The protocol's margin ratio calculation contains a critical flaw that allows users to avoid realizing trading losses by achieving zero asset exposure. When `calcHypotheticalMR()` determines that a position has no asset exposure (`balanceAsset = debtAsset`), it returns the maximum margin ratio without considering the position's actual PnL, effectively treating the position as risk-free regardless of accumulated losses.

Exploit Mechanism:

1. User opens a leveraged position that subsequently moves underwater due to adverse price movements
2. User opens an offsetting trade in the opposite direction to eliminate asset exposure (e.g., if short 1000 vAsset, opens a 1000 vAsset long position)
3. The offsetting trade locks in losses and may incur additional trading costs, worsening an already negative PnL
4. Despite negative PnL, `calcHypotheticalMR()` returns `MMRDecimals` (100% margin ratio) because `positionValue = 0`
5. User can withdraw all collateral since `_checkMR()` passes, leaving the protocol to absorb the bad debt

src/Vault.sol:L183-L192

```
function removeCollateral(uint256 amount, bytes memory unverifiedReport) public nonReentrant {
    IOracleMiddleware(PerpPair(perpPair).oracle()).verifyReportIfNecessary(unverifiedReport);
    require(amount <= userCollateral[_msgSender()], "RC1"); //Error on removeCollateral: Amount exceeds user collateral
    if(amount != userCollateral[_msgSender()]){
        require(amount >= minCollateralMovement, "RC2"); //withdrawal lower than minimum
    }
    require(amount <= totalCollateral, "RC3"); //Error on removeCollateral: Amount exceeds total collateral


    require(_checkMR(amount, _msgSender()), "RC4"); //Error on removeCollateral: MMR check
```

Root Cause:

src/util/UtilMath.sol:L170-L181

```
(uint256 pnl, bool pnlSign) = _calcPnL(
    balanceStable, balanceAsset, debtStable, debtAsset, fundingFee, fundingFeeSign, price, oracleDecimals, perpPair
);

uint256 positionValue = diffAbs(balanceAsset, debtAsset) * price / oracleDecimals;

(uint256 totColl, bool totCollSign) = signedSum(collateral, true, pnl, pnlSign);

//position empty, any collateral is allowed
if (positionValue == 0) {
    return MMRDecimals;
}
```

This design assumes that zero asset exposure equals zero risk, ignoring that users may have accumulated substantial losses through trading activity.

Impact: This vulnerability enables a sophisticated arbitrage strategy where users open separate leveraged positions in both directions. Upon market movement, users can close winning positions through normal procedures to capture profits, while exploiting the zero-exposure flaw on losing positions to withdraw collateral without realizing losses. This asymmetric risk profile allows users to systematically extract value from the protocol by socializing losses while privatizing gains.

5.2 A Liquidator Can Liquidate a Position With Infinitely Bad Price Critical ✓ Fixed

Resolution

Partially fixed in commit f3869fba8db140ed26fbd92cd39793f0679dc749. They have added a check that makes sure that the resulting user position after being liquidated cannot have bad debt, but the check is not well implemented and a check like this does not prevent liquidators from manipulating the price to steal the full users collateral and generating bad debt because of the liquidation discount, therefore creating issue issue 6.1.

The fix in commit 689db8b3a5eced33e1d8744b2c89821f261d58a9 for issue issue 6.1 also fixes this issue.

Description

Similar to another issue (issue 5.3), a liquidator can liquidate any liquidatable position with an infinite profit for themselves and an infinite loss for the liquidated user. Since the liquidation process uses the same logic as regular trades, the liquidator can manipulate the liquidity size to closely match the liquidation size, resulting in an “almost infinite” price.

Recommendation

The recommendation aligns with those from related issues: reconsider the price mechanism for calculating PnL and liquidation price, and implement measures to limit bad debt to reasonable amounts.

5.3 Closing a Position Can Create an Infinite Bad Debt Critical ✓ Fixed

Resolution

Fixed in commit f3869fba8db140ed26fbd92cd39793f0679dc749. An additional check was added to `_closeAndWithdraw` to make sure that a user cannot close his position if this position creates bad debt. Liquidations aren’t affected by this mechanism.

Description

When a user wishes to close their position, they may call either `closeTrade` or `closeAndWithdraw`. Both functions allow the user to close any open position. At the end of `closeTrade`, only the Maintenance Requirement (MR) is checked

src/perpModules/perpTrade.sol:L76-L79

```
require(
    UtilMath.calcMR(user, spotPrice, address(this), getCollateral(user), lastOperationTimestamp) > MMR,
    "OT1"
);
```

In `closeAndWithdraw`, not even the MR is checked, as the position is assumed to be zero and funds are withdrawn.

While these functions ensure that trades cannot execute at a price better than the spot price, they do not prevent trades from executing at extremely unfavorable prices for the trader. For example, if a trader attempts to buy nearly all assets in the liquidity pool, the average price can become arbitrarily large. An attacker can manipulate the trade size to approach the pool’s total assets, effectively driving the price to “near infinity.”

This behavior enables the creation of unbounded bad debt, which is absorbed by the protocol, while liquidity providers receive excessive profits. An attacker could exploit this to drain the protocol.

Example attack scenario:

1. The liquidity pool contains 1000 assets.
2. The attacker uses a flash loan and multiple accounts to orchestrate the attack.
3. Account_1 deposits 1000 assets as liquidity (pool: 2000 assets).
4. Account_2 sells 1000 assets (pool: 3000 assets).

5. Account_3 buys 1999.999999 assets (pool: 1000.000001 assets remain).
6. Account_2, with a -1000 position, buys 1000 assets to close. Purchasing 1000 assets from a pool with 1000.000001 assets results in a "near infinite" price and bad debt, which the protocol accepts.
7. Account_1, as the liquidity provider, receives excessive profit and can drain the Vault.

These actions could theoretically be performed within a single transaction.

Recommendation

The core issue is the lack of PnL consideration when permitting users to close positions. Addressing this is non-trivial: while limiting per-trade losses can prevent infinite prices, it does not fully mitigate the risk, as attackers could still exploit the protocol with smaller, repeated attacks. It is a design decision to determine which trades should be allowed as legitimate stop-losses and which should be restricted to protect protocol solvency. Any trades that result in bad debt are a potential attack vector.

5.4 LP Liquidity and Trader Liquidity Mix Causes Open Interest Corruption Major ✓ Fixed

Resolution

Partially fixed in commit f3869fba8db140ed26fbd92cd39793f0679dc749. They separated the LP balances and debts from the trader balances and debts, and _removeLiquidity now merges the LP balances and debts with the trader ones. However, a new issue issue 6.3 was created because of inconstancies when applying this new accounting system

Description

The `_removeLiquidity()` function converts LP positions to trader balances without updating the protocol's trader exposure accounting. Since the converted position now represents active trader exposure rather than passive LP counterparty exposure, it should be included in `totalTraderExposure` calculations for accurate protocol state tracking.

Impact: This accounting oversight creates incomplete exposure tracking where `totalTraderExposure` underrepresents actual trader positions in the system. The resulting funding rate distortion occurs because funding calculations rely on incorrect exposure data, leading to mispriced risk and improper incentive structures. Subsequent trades operate with a corrupted open interest baseline, creating cascading accounting errors that compound over time and fundamentally compromise the protocol's understanding of total market exposure, making risk assessment unreliable.

Example: In a pool with a stable price of 1, no slippage:

1. LP deposits 1000 vAsset and registers 1000 vAsset debt
2. Trades in the pool change his position to now have 1000 vStable liquidity and 1000 vAsset debt (net -1000 vAsset exposure)
3. LP removes liquidity → becomes trader with 1000 vStable balance and 1000 vAsset debt. `totalTraderExposure` is not updated to reflect this new -1000 vAsset trader position
4. Trader closes position by trading 1000 vStable → 1000 vAsset to cover debt
5. This trade increases `totalTraderExposure` by +1000, despite the trader completely exiting
6. Protocol now shows inflated open interest when the position is fully closed

5.5 Exposure Can Remain Open When Closing the Position Major ✓ Fixed

Resolution

Fixed in commit f3869fba8db140ed26fbd92cd39793f0679dc749. The mechanism to close the exposure in `_closeAndWithdraw` has been redesigned, now the function is centered around exposure and not around debts and balances. It guarantees that any long or short exposure is closed by the end of the function by trading back any remaining exposure.

Description

In the `_closeAndWithdraw` function, various combinations of `pos.balanceAsset`, `pos.debtAsset`, `pos.balanceStable`, and `pos.debtStable` are handled.

src/perpModules/perpTrade.sol:L438-L488

```
if (pos.balanceAsset > pos.debtAsset) {
    unchecked {
        pos.balanceAsset -= pos.debtAsset;
    }
    pos.debtAsset = 0;
    uint256 minTradeReturn = pos.balanceAsset*price/oracleDecimals * (1e5 - maxSlippage)/1e5;
    uint256 tradeReturn = _trade(false, pos.balanceAsset, minTradeReturn, globalLiquidityStable, frontendAddress, user, price)
    emit ExecutedTrade(user, 0, false, pos.balanceAsset, tradeReturn, price, 1);
} else {
    unchecked {
        pos.debtAsset -= pos.balanceAsset;
    }
    pos.balanceAsset = 0;
}

// Repay stable debt
if (pos.balanceStable > pos.debtStable) {
    unchecked {
        pos.balanceStable -= pos.debtStable;
    }
    pos.debtStable = 0;
    if (pos.debtAsset > 0) {

        uint256 inputNeeded = ((
            CurveMath.computeExactAmountInLong(
                pos.debtAsset,
                price,
                oracleDecimals,
                globalLiquidityStable,
                globalLiquidityStable,
                globalLiquidityAsset,
                curveParameters.longCurveParameterA,
                curveParameters.longCurveParameterB,
                1e8
            ) + flatTradingFee) * decimals.tradingFeeDecimals
        ) / (decimals.tradingFeeDecimals - tradingFee);

        uint256 minTradeReturn = inputNeeded*oracleDecimals/price * (1e5 - maxSlippage)/1e5;
        unchecked {
            uint256 tradeReturn = _trade(true, inputNeeded, minTradeReturn, globalLiquidityAsset, frontendAddress, user, price
            emit ExecutedTrade(user, 0, true, inputNeeded, tradeReturn, price, 1);
        }

        require(UtilMath.diffAbs(pos.balanceAsset, pos.debtAsset)*price/oracleDecimals < 1e10, "CW2");
    }
} else {
    unchecked {
        pos.debtStable -= pos.balanceStable;
    }
    pos.balanceStable = 0;
}
```

However, when both `pos.balanceAsset < pos.debtAsset` and `pos.balanceStable < pos.debtStable`, no action is taken, leaving the exposure `pos.balanceAsset - pos.debtAsset` unclosed. This leads to two main issues:

1. The user pays a "theoretical" loss as calculated by the `calcPnL` function, rather than the actual realized loss from closing the position, which should include fees and potentially increased slippage.

src/perpModules/perpTrade.sol:L490-L491

```
// Calculate PnL
(uint256 pnl, bool pnlSign) = calcPnL(user, price);
```

2. The exposure remains open, meaning the system's overall open interest is not properly balanced.

Recommendation

Ensure that all exposures are fully closed when a position is closed.

5.6 Oracle Inverted Price Validity Major ✓ Fixed

Resolution

Fixed in commit f3869fba8db140ed26fbd92cd39793f0679dc749.

Description

The oracle's inverted price validity logic creates a critical DoS vulnerability where any report submission immediately makes prices unavailable until expiration.

Root Cause: Price validity requires expiration to have ALREADY passed

src/CL_oracle_middleware/TWAPOracleMiddleware.sol:L140-L144

```
function getPrice () external view returns (int192 price) {
    require (((lastDecodedPrice != 0) &&
        (lastDecodedValidFromTimestamp >= block.timestamp-maxTimeDelta) &&
        (lastDecodedValidFromTimestamp <= block.timestamp) &&
        (lastDecodedExpiresAt < block.timestamp)),
```

// But new reports can only be submitted when expired if (block.timestamp >= lastDecodedExpiresAt) { verifyReport(unverifiedReport); }

Attack Vector:

1. Wait for expiration window when new reports can be submitted
2. Submit any valid report with future expiration date
3. Price becomes immediately invalid and remains so until new expiration
4. All protocol operations fail when calling getPrice()
5. Repeat at each expiration to maintain permanent DoS

Impact:

- Complete protocol DoS through legitimate report submission
- Self-DoSing system - every new report breaks price availability
- No viable workaround - protocol fundamentally unusable
- Attackers can maintain permanent downtime with minimal cost

5.7 Users Without Collateral May Lose Their PnL Major ✓ Fixed

Resolution

Fixed in commit f3869fba8db140ed26fbd92cd39793f0679dc749. If any of the `userCollateralRatio` is missing, it set the user's collateral ratios to equal the vault's collateral ratios.

Description

When withdrawing collateral from the Vault, it is assumed that the `userCollateralRatio` mapping is not empty. However, if `userCollateralRatio` is empty, the `_removeCollateral` function will still execute successfully:

src/Vault.sol:L220-L240

```
function _removeCollateral(uint256 amount, address user) private returns (uint256[] memory removedCollateral) {
    updateSnapshot();
    removedCollateral = new uint256[](stableCoins.length);
    uint256[] memory newGlobalRatios = new uint256[](stableCoins.length);

    // 1. Compute removed collateral and new global ratios
    bool enoughCollat = _computeCollateralRemoval(amount, user, removedCollateral, newGlobalRatios);

    // 2. Check if new ratios are allowed
    bool goodRatios = newRatiosAllowed(newGlobalRatios, false);

    // 3. Update ratios accordingly
    if (enoughCollat && goodRatios) {
        _updateGlobalRatios(newGlobalRatios);
    } else {
        _updateUserRatios(amount, user, removedCollateral);
    }
    totalCollateral -= amount;
    userCollateral[user] -= amount;
    return removedCollateral;
}
```

In this scenario, the user's collateral is reduced by the specified `amount`, but the `removedCollateral` array—which should contain the amounts for each stablecoin—remains empty. As a result, the funds are considered withdrawn, but no actual stablecoin is transferred, leading to a loss of the `amount` for the user.

A user can have an empty `userCollateralRatio` in two cases:

1. The user never deposited collateral and only received fees.
2. The user withdrew all collateral while still maintaining a position or providing liquidity (e.g., due to positive PnL covering the position). Later, when the user attempts to close everything and withdraw their PnL, the funds will be lost.

Recommendation

Ensure that `userCollateralRatio` is never empty, or default to using `totalCollateralRatio` when it is. This will prevent users from losing funds in these edge cases.

5.8 PnL Can Be Artificially Lowered by Manipulating the AMM Price Major ✓ Fixed

Resolution

Fixed in commit f3869fba8db140ed26fbd92cd39793f0679dc749. A `useSpotPrice` bool has been added to the `calcPnL` function as a parameter where it will determine if the price for the PnL will use spot price or not. MR checks will use spot price while `_updateSnapshots` will use the curve price.

Description

In addition to the concerns raised in issue 5.9 and issue 5.2 , the current method of calculating PnL using the pool price instead of the spot price introduces a significant risk. Even healthy or profitable positions can be immediately classified as "bad debt" if their size approaches the available liquidity in the pool. This is because simulating the immediate closure of a position allows the price to be manipulated to extremely unfavorable levels.

A liquidator can exploit this vulnerability to target any position. While system-wide bad debt may be limited, and draining the entire protocol may not be possible, a liquidator could still steal all the funds of a specific user by liquidating their position under these conditions.

Recommendation

This issue further reinforces the recommendation to use the spot price for PnL and liquidation calculations.

5.9 PnL Calculation Can Fail Due to Lack of Liquidity Major Acknowledged

Resolution

Acknowledged. The Denaria team's comment: "The protocol is not meant to work when the liquidity is extremely low. If you could not close your position due to lack of liquidity your PnL is not defined."

Description

When calculating PnL, the system simulates closing the position using the current state of the liquidity pool:

src/util/UtilMath.sol:L224-L246

```
if(diffAssetSign){
    shortReturn = CurveMath.computeShortReturn( diffAsset,
                                                 price,
                                                 oracleDecimals,
                                                 getTotalLiquidityStable(perpPair),
                                                 getTotalLiquidityStable(perpPair),
                                                 getTotalLiquidityAsset(perpPair),
                                                 sA,
                                                 sB,
                                                 1e8);
}
else{
    require(diffAsset <= getTotalLiquidityAsset(perpPair), "PNL1"); //Requiring more asset then in the pool to exit. Cannot exi
    shortReturn = CurveMath.computeExactAmountInLong(diffAsset,
                                                 price,
                                                 oracleDecimals,
                                                 getTotalLiquidityStable(perpPair),
                                                 getTotalLiquidityStable(perpPair),
                                                 getTotalLiquidityAsset(perpPair),
                                                 lA,
                                                 lB,
                                                 1e8);
}
```

Within this logic, a `require` statement ensures there is sufficient liquidity for the trade. However, this check is only applied in one trade direction, not both. More importantly, if there is insufficient liquidity, the PnL calculation will revert.

The protocol is intended to function even when liquidity is depleted. In such scenarios, it becomes impossible to calculate the closing price, which in turn prevents the liquidation of positions. This limitation can result in unresolvable positions and potential bad debt for the protocol.

Recommendation

PnL calculation should always be possible, regardless of liquidity conditions. For example, the system could use the spot price to simulate closing the position, ensuring that liquidation and debt resolution mechanisms remain functional.

5.10 Removing Collateral for Liquidated Position Is Unnecessary Medium Acknowledged

Resolution

Acknowledged. The Denaria team's comment: "While unnecessary we deem that this addition provides a significant benefit to the ux. The considered stablecoins for the protocol are being carefully chosen to not cause issues, such as infinite gas consumption."

Description

When liquidating a user's position, the collateral is removed from the vault:

src/perpModules/perpLiquidation.sol:L105-L108

```
if(fraction == decimals.liquidationDecimals){
    _closeAndWithdraw(1e5, 1e10, _msgSender(), user);
    IVault(vault).removeAllCollateralForUser(user);
}
```

That is unnecessary and carries an additional risk by performing more actions, running out of gas, needing more code, and creating reentrancy attack vectors. In addition, the `removeAllCollateralForUser` is not protected by the reentrancy guard.

The main danger is that the mechanism that prevents ERC20 transfers from reverting is not bulletproof. For example, the calls can still run out of gas:

src/Vault.sol:L249-L257

```
if (removedCollateral[i] > 0){
    (bool success, ) = address(stableCoins[i].stableCoin).call(
        abi.encodeWithSelector(IERC20.transfer.selector, user, removedCollateral[i])
    );
    if(!success){
        IERC20(address(stableCoins[i].stableCoin)).approve(lostAndFound, removedCollateral[i]);
        ILostAndFound(lostAndFound).depositLostFunds(user, address(stableCoins[i].stableCoin), removedCollateral[i]);
        emit BlockedCollateralRemoval(address(stableCoins[i].stableCoin), user, removedCollateral[i]);
    }
```

Recommendation

Don’t remove collateral for the user in case of liquidations.

5.11 Bad Debt in a System Can Remain Unreported Medium Acknowledged

Resolution

Acknowledged. The Denaria team’s comment: “This is true due to the lack of automatic updates. When the user gets liquidated his bad debt gets accounted for. This is only an issue if liquidations fail.”

Description

Bad debt is only reported in the `_closeAndWithdraw` function:

src/perpModules/perpTrade.sol:L498-L501

```
if (getCollateral(user) < pnl && !pnlSign){
    (insuranceFund, insuranceFundSign) = UtilMath.signedSum(insuranceFund, insuranceFundSign, pnl - getCollateral(user), false
}
IVault(vault).addPnlToCollateral(user, pnl, pnlSign);
```

However, a user can close their position using `closeTrade` and never call `closeAndWithdraw`. Since there are no funds to retrieve, there is no incentive to call `closeAndWithdraw`. As a result, the system may have bad debt that is never recognized or accounted for.

Recommendation

Ensure that bad debt is always acknowledged, regardless of how the position is closed.

5.12 Withdrawal Race Condition in Case of Bad Debt Medium Acknowledged

Resolution

Acknowledged. The Denaria team’s comment: “This is true for most protocols, we decided not to apply penalties to withdrawals.”

Description

When bad debt occurs, the system treats it as a (temporary) loss. Users are able to withdraw their collateral without penalties, resulting in an inconsistency between the Vault’s token balances and the collateral amounts attributed to users. If many users begin withdrawing funds during periods of increasing bad debt, the Vault may eventually be depleted of funds, while some users are still shown as having collateral in the system. Ultimately, these remaining users will bear the loss caused by the bad debt.

The protocol includes an insurance fund, funded by a small fee on every trade, to help cover bad debt and maintain the system’s balance. While this is a desirable behavior, there may still be scenarios where a spike in bad debt triggers a wave of withdrawals, leading to a withdrawal spiral and a race condition among users.

Recommendation

As an additional safeguard alongside the insurance fund, consider implementing a mechanism to proportionally penalize all users withdrawing from the system when bad debt is present. This would help distribute losses more fairly and discourage withdrawal spirals during periods of insolvency.

5.13 Withdraw/Deposit vs Decrease/Increase for Collateral Ratios Medium Acknowledged

Resolution

Acknowledged. The Denaria team’s comment: “The current system does not cause issues to the accounting, it may have different pros and cons, but we decided to stick with our current implementation.”

Description

In the Vault there is a limit how much the ratios of stablecoins can increase or decrease as the result of withdrawals:

src/Vault.sol:L461-L475

```
function newRatiosAllowed(uint256[] memory newRatios, bool isDeposit) private view returns (bool allowed) {
    uint256 threshold;
    for (uint256 i; i < newRatios.length; i++) {
        if (isDeposit) {
            threshold = stableCoins[i].depositRatioThreshold;
        } else {
            threshold = stableCoins[i].withdrawalRatioThreshold;
        }

        if (UtilMath.diffAbs(ratiosSnapshot[i], newRatios[i]) > threshold) {
            return false;
        }
    }
    return true;
}
```

It's called `depositRatioThreshold` and `withdrawalRatioThreshold`.

While the idea seems promising, the increase in one stablecoin means a corresponding decrease in others. While both are happening during the same deposit or withdrawal.

Recommendation

Consider changing deposit/withdrawal thresholds to increase/decrease thresholds.

5.14 Frontend Fees Are Paid by Liquidity Providers Medium ✓ Fixed

Resolution

Fixed in f3869fba8db140ed26fbd92cd39793f0679dc749. `adjSize` was modified to contemplate the frontend fee when the `frontendAddress` is `0`.

Description

During the "buy" trade, frontend fees are not accounted for correctly when the frontend address is zero. The current calculation causes LP providers to pay the frontend fees instead of the trader:

src/perpModules/perpTrade.sol:L321-L325

```
if (direction) {
    unchecked {
        uint256 adjSize = size - tradingFeeAmount*(decimals.feeFractionsDecimals - feeLP) / decimals.feeFractionsDecimals;
        uint256 aY = (globalSharesStable * adjSize * SafeCast.toUint256(decimals.liquidityMDecimals))
            / (globalSharesAsset * stableLiq);
```

Recommendation

The adjusted size should be increased by the frontend fees when the frontend address is zero.

5.15 Curve State Accumulation (Dy0/Dx0) Issues Medium Partially Addressed

Resolution

Partially addressed: Although the `closeAndWithdraw` function was modified to consider the accumulators, the Denaria team has decided to keep the overall structure, as they deem its contribution to preventing splitting a trade into smaller ones from being profitable to be relevant enough to justify the added complexity.

Description

The price formula in the protocol is designed so that every new trade starts with the spot price, and slippage increases with trade size. The downside is that a large trade can be split into multiple smaller trades to achieve a better average price. To mitigate this, the protocol implements an additional mechanism: the curve does not recompute after every trade. If the curve remains unchanged, follow-up trades do not start at the spot price but continue from where the previous trade ended, like in a classic AMM, as if multiple trades in the same direction were a single large trade. The curve recomputes either over time, or when a trade in the opposite direction occurs, or when liquidity is added/removed.

While this mechanism makes trade splitting more difficult, it does not fully solve the issue and introduces additional problems:

- It is still easy to reset the curve by making a small trade in the opposite direction or by adding a small amount of liquidity. While this incurs extra fees, for large trades it can still be worthwhile.
- The mechanism adds significant complexity. To simulate non-recomputing of the curve, previous trades must be stored and accumulated, then accounted for in subsequent trades. This is done via accumulating `dx0` and `dy0` values, which are added/subtracted from relevant calculations. These calculations are complex and error-prone.

src/perpModules/perpTrade.sol:L214-L227

```
    tradeReturn = CurveMath.computeLongReturn(
        size - tradingFeeAmount + dy0,
        spotPrice,
        oracleDecimals,
        initialGuess,
        stableLiq - dy0,
        assetLiq + dx0,
        curveParameters.longCurveParameterA,
        curveParameters.longCurveParameterB,
        1e8
    ) - dx0;
    dy0 += size - tradingFeeAmount;
}
dx0 += tradeReturn;
```

src/perpModules/perpTrade.sol:L246-L260

```
shortTotalTradeReturn = CurveMath.computeShortReturn(
    size + dx0,
    spotPrice,
    oracleDecimals,
    initialGuess + dy0,
    stableLiq + dy0,
    assetLiq - dx0,
    curveParameters.shortCurveParameterA,
    curveParameters.shortCurveParameterB,
    1e8
) - dy0;

//Might miss LP fees, but even if it does the effect of this is minimal (impact of fees on liquidity and thus slippage should be
dx0 += size;
dy0 += shortTotalTradeReturn;
```

Examples of issues include:

- To accurately simulate multiple trades as a single trade, the curve should return to the exact parameters it had at the start of the first trade. However, this is not achieved because fees are not accounted for in the calculation. The curve will be similar, but not identical, to what it should be. While the difference (fees) may seem small, a malicious actor manipulating liquidity size could potentially exploit this.
- The `dy0` / `dx0` calculation is not performed when predicting input amounts for the `closeAndWithdraw` function, but is performed during the actual trade, leading to inaccurate results or failed transactions.

src/perpModules/perpTrade.sol:L461-L481

```
uint256 inputNeeded = ((
    CurveMath.computeExactAmountInLong(
        pos.debtAsset,
        price,
        oracleDecimals,
        globalLiquidityStable,
        globalLiquidityStable,
        globalLiquidityAsset,
        curveParameters.longCurveParameterA,
        curveParameters.longCurveParameterB,
        1e8
    ) + flatTradingFee) * decimals.tradingFeeDecimals
) / (decimals.tradingFeeDecimals - tradingFee);

uint256 minTradeReturn = inputNeeded*oracleDecimals/price * (1e5 - maxSlippage)/1e5;
unchecked {
    uint256 tradeReturn = _trade(true, inputNeeded, minTradeReturn, globalLiquidityAsset, frontendAddress, user, price);
    emit ExecutedTrade(user, 0, true, inputNeeded, tradeReturn, price, 1);
}

require(UtilMath.diffAbs(pos.balanceAsset, pos.debtAsset)*price/oracleDecimals < 1e10, "CW2");
```

- The `dy0` / `dx0` calculation is not considered in liquidation or PnL calculations. While this behavior may be desirable, it introduces inconsistency and additional complexity in the code.

Recommendation

Reconsider the `dy0` / `dx0` mechanism.

5.16 Liquidity Provider Can Have Higher Leverage Than Allowed by the System Medium Acknowledged

Resolution

Acknowledged. The Denaria team’s comment: “The maximum leverage we imposed on LP deposits is not a hard limit on what LPs can achieve. While it is vital that there is a boundary on LPs leverage to prevent them from depositing infinite liquidity, their positions are in general less risky for the system, as their exposure is usually far from their maximum potential exposure, against which we perform this leverage check. Our leverage check makes it so that you cannot instantly achieve a higher leverage as a direct result of your actions, but you can only do so through losing part of your collateral due to a negative PnL. If a user is dedicated enough to achieve a better capital efficiency we do not prevent them from doing so, at the cost of having a much riskier position, which is more susceptible to liquidations. In fact, any position’s leverage is still bound by the MMR, which is checked against all position’s MR for liquidations in the same way. The fact that LP maximum leverage is lower than Trader’s maximum leverage is due to the fact that we want to provide a “safer” experience for casual LPs, further reducing the risk of their liquidation.”

Description

There are two mechanisms intended to ensure sufficient collateral in the Vault for each user:

- **Margin Ratio (MR) Limit** – Calculates the ratio of a user's exposure to their collateral. If this ratio falls below a certain threshold, the position is subject to liquidation.

src/util/UtilMath.sol:L186-L187

```
marginRatio = totColl * MMRDecimals
    / (positionValue);
```

src/perpModules/perpTrade.sol:L76-L79

```
require(
    UtilMath.calcMR(user, spotPrice, address(this), getCollateral(user), lastOperationTimestamp) > MMR,
    "OT1"
);
```

- **Maximum LP Leverage** – Sets a cap on leverage for liquidity provision. For example, with a maximum leverage of 10, a user must have at least $100 collateral to provide $1000 in liquidity.

src/perpModules/perpLiquidity.sol:L81-L85

```
VirtualTraderPosition storage position = userVirtualTraderPosition[sender];
uint256 debtStable = position.debtStable > position.balanceStable ? position.debtStable-position.balanceStable : 0;
uint256 debtAsset = position.debtAsset > position.balanceAsset ? position.debtAsset-position.balanceAsset : 0;

require(debtStable + debtAsset*spotPrice/oracleDecimals <= getCollateral(sender)*maxLpLeverage, "L3"); //To deposit liquidity y
```

However, there are several issues with these mechanisms:

1. **Lack of Synchronization:** These two mechanisms operate independently and are not coordinated. Users face two sources of risk: open positions and liquidity that can become exposure over time. Ideally, both should be independently collateralized, but currently, the same collateral is used to cover both, effectively "double-counting" the collateral.
2. **Infrequent Leverage Checks:** The liquidity leverage check is only enforced at the time of providing liquidity and when collateral is withdrawn. This means that in between these events, a user's liquidity leverage can exceed the intended maximum without detection by the system. For example, a user may use their PnL as collateral to provide liquidity at maximum leverage. If the PnL then decreases, the user can end up with a significant liquidity in the pool and very low collateral backing it.

Recommendation

Ideally, liquidity and position exposure should each require their own dedicated collateral and be tracked continuously. If there is insufficient collateral to support the liquidity, the protocol should automatically reduce or remove the user's liquidity position.

5.17 Auto Close Is Not Disabled After Execution Minor ✓ Fixed

Resolution

Fixed in commit f3869fba8db140ed26fbd92cd39793f0679dc749.

Description

The auto close allows others to close the user's position under certain conditions:

src/perpModules/perpAutoClose.sol:L44-L56

```
function autoCloseUserPosition(address user, address frontendAddress, bytes memory unverifiedReport) external nonReentrant {
    IOracleMiddleware(oracle).verifyReportIfNecessary(unverifiedReport);
    require(autoCloseUsersData[user].authorized, "A2");
    (uint256 userPnL, bool userPnLSign) = calcPnL(user, getPrice());
    if (userPnLSign){
        require(autoCloseUsersData[user].profitTh != 0 && userPnL >= autoCloseUsersData[user].profitTh, "A3");
    }
    else{
        require(autoCloseUsersData[user].lossTh != 0 && userPnL >= autoCloseUsersData[user].lossTh, "A4");
    }
    _applyAutoClosureFees(_msgSender(), user);
    _closeAndWithdraw(autoCloseUsersData[user].maxSlippage, autoCloseUsersData[user].maxLiqFee, frontendAddress, user);
}
```

The issue is that it's not deleted afterward. Enabling this behavior again later, unintentionally.

Recommendation

Delete the `autoCloseUsersData` after executing.

5.18 Fee Timing Creates Directional Trading Bias Minor Acknowledged

Resolution

Acknowledged. The Denaria team's comment: "We are aware of such asymmetry. We performed some computations to check the relevance of this discrepancy and we found that the magnitude of the difference is proportional to the product of slippage and fees, so it can be neglected without causing significant issues. "

The protocol applies different fee timing for long and short trades, creating systematic bias that favors short trading in high slippage environments and disadvantages long positions.

Long trades calculate fees on the input amount before executing trades on reduced liquidity, while short trades execute on full input amounts and calculate fees from the resulting output. This asymmetry means long trades suffer from both upfront fee reduction and subsequent slippage on the smaller traded amount, whereas short trades execute with full input amounts and only pay fees on the slippage-affected returns. In high slippage scenarios, this creates a meaningful economic advantage for short positions since they pay fees on slippage-reduced returns rather than full input amounts like longs.

Impact: The inconsistent fee timing between long and short trades creates a slight asymmetry in fee treatment, with short positions receiving marginally lower fees during high slippage scenarios. While the economic impact is minimal under normal market conditions, the asymmetric structure adds system complexity and creates theoretically unequal treatment between trading directions.

5.19 Lack of Flexibility Withdrawing PNL Minor ✓ Fixed

Resolution

Fixed in commit f3869fba8db140ed26fbd92cd39793f0679dc749. A new `realizePnL` function was added which applies the PnL to the vault's collateral.

Description

The only place where PnL can be attributed to the collateral in the Vault is the `_closeAndWithdraw` function:

src/perpModules/perpTrade.sol:L501

```
IVault(vault).addPnlToCollateral(user, pnl, pnlSign);
```

src/Vault.sol:L326-L338

```
function addPnlToCollateral(address user, uint256 pnl, bool pnlSign) external onlyPerpPair {

    if (pnlSign) {
        userCollateral[user] += pnl;
    } else {
        if(userCollateral[user] >= pnl){
            userCollateral[user] -= pnl;
        }
        else{
            userCollateral[user] = 0;
        }
    }
}
```

So the only way to retrieve PnL is by closing the position and withdrawing from the system fully. This approach may not be desirable for many users. There are some situations where users would want to withdraw part of their PnL when rebalancing their position.

Examples

1. When a user has a large position with profit, it would make sense to withdraw part of the profit, not needed to back up the position, to put this profit to work somewhere else. Currently, the only way to do this is to close the full position and then reopen it again. This is a very costly solution as it would require paying fees for these big trades plus double the slippage. So while this behavior is usually achievable, it's very costly for no real reason.
2. When there is not enough liquidity in the pool to close the position, even if the residual position is quite small, the user's PnL can be huge but absolutely impossible to withdraw.

Recommendation

Consider adding more flexibility in transferring user's PnL to the Vault.

5.20 OpenTrade Mechanism Is Not Consistent Onchain ✓ Fixed

Resolution

Fixed in commit f3869fba8db140ed26fbd92cd39793f0679dc749.

Description

There are 2 places where `openTrades` is used:

- Opening trade:

src/perpModules/perpTrade.sol:L404-L408

```
function logTradePosition(address user) private returns(bytes32 id){
    cumulativeTradeCount++;
    id = keccak256(abi.encodePacked(user, cumulativeTradeCount, address(this), block.chainid));
    openTrades[id] = user;
}
```

- Closing trade:

src/perpModules/perpTrade.sol:L138

```
delete openTrades[tradeID];
```

This mechanism is not used in any meaningful way on-chain. It also does not account for liquidations, auto trades and `closeAndWithdraw` function.

Additionally, in `openTrade` function the trade is added to the `openTrades` inside the parameter calculation of the event function, which is bad code style:

src/perpModules/perpTrade.sol:L82-L90

```
emit ExecutedTrade(
    user,
    logTradePosition(user),
    direction,
    size,
    tradeReturn,
    spotPrice,
    leverage
);
```

6 Fix Review Findings

6.1 Protocol Value Drainage Through Liquidation Manipulation Critical ✓ Fixed

Resolution

This issue was fixed in commit f0b4ea5bad514459d9dfdbc3bc05417219c9864d. The Denaria team added an Exponential Moving Average (EMA) of the slippage registered for short and long trades, updated up to once per block. When liquidating the system checks if the slippage during the liquidation would exceed this average by a margin (e.g. 3x average). If that's the case it deems there's an exploit being carried on and it uses spot price for the liquidation.

Description

This issue was found during the fix review, and was a result of the attempt to fix issue issue 5.2.

The protocol implements a bad debt protection mechanism in the liquidation system to that attempts to prevent insolvency by checking if curve manipulation would push a user's losses beyond their available collateral. When this threshold is reached, the system falls back to spot pricing under the assumption that fair market rates will prevent bad debt generation.

However, this protection mechanism contains fundamental design flaws that create systematic exploitation opportunities rather than genuine risk mitigation. The protection operates on a **per-transaction basis without considering the cumulative impact of liquidation discounts** or the sustainability of positions after partial liquidations, creating windows where attackers can extract maximum value while technically staying within the "safe" parameters.

Root Cause

The core vulnerability stems from the protocol's reliance on liquidation discounts calculated as percentages of manipulated prices combined with insufficient forward-looking solvency checks. Attackers can exploit this by using micro-liquidations with extreme price manipulation to drain a user's effective collateral up to the exact threshold where bad debt protection would trigger, leaving positions at precisely 0% margin ratio.

This setup enables a **two-phase extraction strategy** where:

1. The first liquidation extracts all available collateral through curve manipulation
2. The second liquidation applies maximum discounts to the remaining position at spot prices, generating substantial bad debt that gets socialized across the protocol

The combination of legitimate maximum leverage positioning, strategic collateral withdrawal, and systematic liquidation exploitation creates a **risk-free profit mechanism** that fundamentally breaks the protocol's economic security model.

Example

The strategy combines legitimate maximum leverage usage with collateral withdrawal to the MMR limit, followed by systematic liquidation exploitation. The massive bad debt gets socialized across the protocol's LPs and insurance fund, representing a complete breakdown of the risk management system.

Phase 1: Enhanced Position Setup with Maximum Leverage Extraction

Reference: Maximum allowed leverage is 15x, MMR = 3.8%

Attacker creates two accounts and maximizes leverage through collateral withdrawal:

Account	Position Type	Asset Debt	Stable Balance	Initial Collateral	Withdrawn	Remaining Collateral	Effective Leverage	Entry Price
Main Account	Short	15,000	15,000	1,000	430	570	26.3x	1.00
Liquidator Account	-	-	-	-	-	-	-	-

Phase 2: Asymmetric Outcome Strategy

Scenario A: Price Decreases (Natural Profit)

- Short position generates trading profits proportional to 15x leverage
- Close position normally, retain profits + collateral

Scenario B: Price Increases (Exploit Activation)

- Price Movement: 1.00 → 1.001 (+0.1%)
- Main Account Impact: PnL = -15, Effective Collateral = 555
- Status: Position becomes liquidatable (MR = 3.696% < 3.8% MMR)

Phase 3: Double Liquidation Extraction Attack

Step 1: Curve Manipulation Setup

1. Manipulate curve to inflate asset price from 1.001 to 555
2. Execute attack sequence
3. Restore curve to fair pricing (minimize net manipulation cost)

Step 2: First Liquidation (Micro-extraction)

- Target: 1 asset (0.007% of position)
- Manipulated Price: 555 stables per asset (vs 1.001 fair value)
- Liquidation Discount: ~2% (position just became liquidatable)
- User Payment: 555 × 1.02 = 566.1 stables

User Position After First Liquidation:

- Asset Debt: 14,999
- Stable Balance: 14,433.9 (15,000 - 566.1)
- Effective Collateral: 0 (position now insolvent)

Step 3: Second Liquidation (Full Remaining Position)

- Remaining Position: 14,999 asset debt
- User Margin Ratio: 0% (eligible for maximum 10% discount)
- Liquidation: 14,999 assets at spot price + 10% maximum discount
- User Payment: 14,999 × 1.001 × 1.10 = 16,515.4 stables

Phase 4: Economic Analysis

Total User Payments: 566.1 + 16,515.4 = 17,081.5 stables

User's Available Resources:

- Stable balance after first liquidation: 14,433.9 stables
- Remaining vault collateral: 0 stables (already extracted through first liquidation)
- Total available: 14,433.9 stables

Bad Debt Generated: 17,081.5 - 14,433.9 = 2,647.6 stables

Liquidator's Final Position:

- Asset Debt: 15,000 (acquired entire user position)
- Stables Received: 17,081.5
- Current Market Value: 15,000 × 1.001 = 15,015.0
- Gross Profit: 17,081.5 - 15,015.0 = 2,066.5 stables

Attacker's Total Gains:

- Liquidation profit: 2,066.5 stables

Attack Economics:

- Initial Investment: 1,000 collateral
- Net Profit: 2,066.5 - 1,000 = 1,066.5 stables (106.65% ROI)
- Bad Debt Generated: 1,066 stables (socialized to protocol)

This enhanced attack achieves 106% risk-free returns while generating over 1,066 stables of bad debt per 1,000 invested.

6.2 Inconsistent LP Debt Transfer Allows Users to Avoid Debt Obligations Major ✓ Fixed

Resolution

Fixed in commit 689db8b3a5eced33e1d8744b2c89821f261d58a9. `_closeAndWithdraw` now removes liquidity and applies debts if there is any LP balance **or debt** pending to be closed.

Description

In the new accounting system applied during the fixes of issue issue 5.4, the `_closeAndWithdraw` function after removing the LP position liquidity and applying it to the trader position, it also applies the LP position debts to the trader position, making sure that the LP position accounting is completely applied to the trader position accounting.

However, other parts of the code call `_removeLiquidity` without applying the LP position debts. For instance, the `liquidate` function calls the `_removeLiquidity` function and doesn't apply the debts after removing the liquidity. This reduces the `stableLiquidity` and `assetLiquidity` to zero, therefore the system will not longer consider that the user has an LP position. When the `_closeAndWithdraw` function is called later, it will skip the application of the LP position debts since the LP position no longer has balances.

Additionally, a user that wanted to close and withdraw could avoid paying his debts by first calling the `removeLiquidity` external function before calling the `closeAndWithdraw` function. The external function also doesn't apply the LP position debts while reducing the `stableLiquidity` and `assetLiquidity` to zero, creating the same kind of vulnerability as in previous example.

We believe these mechanisms could be taken advantage of to create zero risk strategies, although the time limited nature of the fix review doesn't allow us to fully articulate the attack.

6.3 `addLiquidity` Doesn't Consider Trader Position Leverage Medium ✓ Fixed

Description

This issue was found during the fix review of commit f3869fba8db140ed26fbd92cd39793f0679dc749. The new accounting system separates liquidity provider accounting from trader accounting.

While applying this new accounting system, the `addLiquidity` doesn't take into consideration the trader accounting, therefore, a user could open a position with leverage over what would be normally expected by first opening a trader position at max trader leverage, and then adding liquidity to the system at max LP leverage.

Appendix 1 - Files in Scope

This review covered the following files:

File	SHA-1 hash
src/CL_oracle_middleware/TWAPOracleMiddleware.sol	`981756b913fa8cf0bf615c77eb92c0f17578152c`
src/LostAndFound.sol	`f7b582f44415d7885fd1526e8fbd6dfee9381f65`
src/PerpPair.sol	`530c879a1adbad9883bb4356107b53243ff89be1`
src/Vault.sol	`12809373125e2c329000376bcac25ade800a35e8`
src/manager/FeeManager.sol	`9f752c56dccadbdba7dbdbe5e4158329072b6ab1`
src/manager/multiCallManager.sol	`86079830544b546571aa38ec26d523f0c1bbb040`
src/perpModules/internalPerpLogic.sol	`f06ce067cbd80218d4aef8b13064498ff8e78fbc`
src/perpModules/perpAutoClose.sol	`d55f7fcd3620bf1a97a08de2cf579894091b8383`
src/perpModules/perpConfig.sol	`ae6e3be0fd51ddb81615eb37c65ba18ac03eb389`
src/perpModules/perpFunding.sol	`0a52090312a0ae7db55be17096111c2afd603cbc`
src/perpModules/perpLiquidation.sol	`26fb1cf06e7e7a34e00f39123b5ce7c051e7cc25`
src/perpModules/perpLiquidity.sol	`aaae6c14b3135da28dc678e4b09ea5109f64f08d`
src/perpModules/perpTrade.sol	`a6dcd6a4d2546f73581098e8f86178b026f14f51`
src/storage/PerpStorage.sol	`ce488927475874e5435d2da09111e6a4cf193293`
src/util/MatrixMath.sol	`c29a04f56dec50b73816deeb02263abc2d2158d1`
src/util/UtilMath.sol	`7b9dd3be9dc12cfd04395167f9a308c8b7aa0567`

Appendix 2 - Disclosure

A.2.1 Purpose of Reports

The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of code and only the code we note as being within the scope of our review within this report. Any Solidity code itself presents unique and unquantifiable risks as the Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond specified code that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. In some instances, we may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

A.2.2 Links to Other Web Sites from This Web Site

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Consensys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that Consensys and CD are not responsible for the content or operation of such Web sites, and that Consensys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that Consensys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. Consensys and CD assumes no responsibility for the use of third-party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

A.2.3 Timeliness of Content

The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice unless indicated otherwise, by Consensys and CD.