CANTINA

Mellow Finance

Security Review

Cantina Managed review by:

Saw-Mon and Natalie, Lead Security Researcher

Deadrosesxyz, Security Researcher

Kaden, Security Researcher

Akshay Srivastav, Associate Security Researcher

May 20, 2024

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

Mellow Protocol is a permissionless vaults ecosystem for capital efficiency, creating the future of optimal cross-protocol multi-token liquidity allocation.

From Apr 8th to Apr 19th the Cantina team conducted a review of mellow-alm-toolkit on commit hash 8413fc09. The team identified a total of **44** issues in the following risk categories:

- Critical Risk: 2
- High Risk: 1
- Medium Risk: 8
- Low Risk: 19
- Gas Optimizations: 4
- Informational: 10

3 Findings

3.1 Critical Risk

3.1.1 Lack of input validation on `callbackParams.gauge` allows for theft of positions

Severity: Critical Risk

Context: VeloAmmModule.sol#L39

Description: The `gauge` parameter is nowhere verified to be an actual Velodrome gauge. This would allow for an attacker to steal all positions which are up for rebalancing.

1. Attacker deposits position in `Core.sol` and sets `callbackParams.gauge` to his own 'fake' `gauge`.
2. Since the gauge is owned by the attacker, the attacker can immediately get back the NFT.
3. Attacker calls `rebalance` on a victim's position. The position with which it will be rebalanced is the attacker's position from above.
4. The attacker then calls `setPositionParams` and sets `callbackParams.gauge` to the victim's gauge.
5. The attacker can then call `withdraw` and get the position from the gauge.

Recommendation: Verify that `callbackParams.gauge` is a Velodrome gauge.

Mellow: Fixed in commit 736eef90.

Cantina: Fixed in commit 736eef90. `VeloAmmModule` now verifies that `callbackParams.gauge` is a valid Velodrome gauge.

Note: it does not however verify that it is the position's corresponding gauge. Could be used by users to not allow for their position to be rebalanced (by calling `setPositionParams` on their position and changing the gauge). Could be used by `LpWrapper` admins to disable deposits/withdrawals indefinitely.

3.1.2 If `operatorFlag == false`, attacker can steal all NFTs within the contract.

Severity: Critical Risk

Context: Core.sol#L169

Description: For the attack, the attacker will need to first deploy a ERC777-like custom token and then a Velodrome pool with it. Let's say victim's position is in `WETH/USDC`. Victim's position `id = 1`.

1. Attacker creates a NFT for his own `ERC777/WETH` token pool (`id 2`) and also a dust position in the `USDC/WETH` pool (`id 3`).
2. Attacker deposits both NFTs in `Core.sol`.
3. Attacker rebalances the 3 positions, inputs them in the following order: Victim's `USDC/WETH`, `ERC777/WETH`, Attacker's `USDC/WETH` (`id 1`, `id 2`, `id 3`).
4. When callback is done, within the callback the user makes a swap, within their own `ERC777/WETH` pool, so some fees are accrued.
5. The user increases the liquidity of `id 3` enough, so there's enough liquidity to bypass the `minLiquidity` check of `id 1`.
6. Remember that the first position is the victim's. The returned id to be rebalanced with will be `id 3` (the attacker's `USDC/WETH` position).
7. Then the position in the `ERC777/WETH` pool will be rebalanced. Upon transferring it, any fees that have been accrued will be sent to the callback. Remember that we have purposefully made some fees accrue, allowing us to 'steal' the transaction here in our ERC777-hook.
8. Now that we've stolen the transaction, we call `withdraw` on the attacker's position (`id 3`). (the rebalancing has not yet finished and it believes `id 3` still belongs to the attacker).
9. Rebalancing continues.
10. We now deposit a dust position to rebalance attacker's `USDC/WETH` position.

In the end, we've stolen the victim's position and their position is tied to an NFT which is not within the contract, nor within the Gauge.

Recommendation: Keep the operator flag up, add `nonReentrant` modifier to `withdraw` and do not allow for empty `callbackParams`.

Mellow: Fixed in commit 736eef90.

Cantina: Fixed in commit 736eef90. Contract now has `nonReentrant` modifiers and does not allow for empty `callback` params.

3.2 High Risk

3.2.1 `getOraclePrice` is prone to manipulation

Severity: High Risk

Context: VeloOracle.sol#L70

Description: `getOraclePrice` is used to retrieve the latest price from a Velodrome pool. There's a faulty assumption, that if the latest observation's timestamp is in the past, the `slot0` price can't have been manipulated, thus it is returned with no extra checks made. The problem is that Velodrome pools write observations once every 15 seconds, therefore allowing for a manipulated price with an outdated observation.

```
if (block.timestamp != blockTimestamp)
    return (spotSqrtPriceX96, spotTick);
```

Recommendation: Use the latest observation price only if more than 15 seconds have passed since.

Mellow: Fixed in commit 736eef90.

Cantina Managed: Fixed in commit 736eef90. Oracle now always returns spot price. It's important for external protocols to not simply rely on the price returned here, but rather also make a call to `ensureNoMEV`.

3.3 Medium Risk

3.3.1 `RebalanceParams.callback` can steal accrued fees of liquidity NFTs during rebalance

Severity: Medium Risk

Context: Core.sol#L221-L223

Description: During the rebalancing process the liquidity NFTs are transferred to the `RebalanceParams.callback` address.

In case the `CallbackParams` of a user's managed position are empty (a scenario when the user just want rebalancing without gauge deposits) then it would be possible for `RebalanceParams.callback` address to claim the accrued fees of that NFT at the moment when it receives the NFT for rebalancing. Essentially stealing the accrued fees of the user.

Recommendation: Similar to these liquidity checks (Core.sol#L236), also consider validating the accrued fees to the NFT.

Mellow: Fixed in commit 736eef90.

Cantina Managed: Issue has been fixed by not supporting empty `CallbackParams`.

3.3.2 `LpWrapper` deposits and withdraws will be bricked if Velodrome gauge is killed.

Severity: Medium Risk

Context: LpWrapper.sol#L255

Description: Within the `LpWrapper` contract, all withdraws make a `withdraw` and `deposit` back in `Core.sol`. The problem is that in case the corresponding Velodrome gauge is killed, it will not allow for any new deposits to happen. This would lead to all funds within the `LpWrapper` forever stuck.

Recommendation: Consider adding an emergency withdraw function.

Mellow: Fixed in commit 736eef90.

Cantina Managed: The attempt to withdraw from the gauge is skipped at VeloAmmModule.sol#L146.

3.3.3 Usage of a TWAP price instead of the exact current `sqrtPriceX96` will lead to wrong calculation of token amounts within the position

Severity: Medium Risk

Context: LpWrapper.sol#L97

Description: When depositing in the `LpWrapper` `getAmountsForLiquidity` is called in order to calculate the the exact `token0` and `token1` amounts within the position. The problem is that the `sqrtPriceX96` used is not the current one, but rather the TWAP value returned from the oracle. This would lead to inaccurate token amounts calculated and users depositing less than expected and at a different ratio than expected.

Recommendation: Use the `slot0`'s `sqrtPriceX96`.

Mellow: Fixed in commit 52c0aaf0.

Cantina Managed: Fixed.

3.3.4 When depositing a position into a gauge, accrued fees are sent to `Core.sol`

Severity: Medium Risk

Context: VeloAmmModule.sol#L185

Description: When depositing a position into a gauge, `collect` is invoked and it transfers the accrued fees to the `msg.sender`. The problem is that in this case, the `msg.sender` is `Core.sol`. If the user's position had accrued any fees prior to the deposit, they'll be left within the contract, up until another user skims them:

```solidity
function deposit(uint256 tokenId) external override nonReentrant {
    require(nft.ownerOf(tokenId) == msg.sender, "NA");
    require(voter.isAlive(address(this)), "GK");
    (,, address _token0, address _token1, int24 _tickSpacing, int24 tickLower, int24 tickUpper,,,,,) =
        nft.positions(tokenId);
    require(token0 == _token0 && token1 == _token1 && tickSpacing == _tickSpacing, "PM");

    // trigger update on staked position so NFT will be in sync with the pool
    nft.collect(
        INonfungiblePositionManager.CollectParams({
            tokenId: tokenId,
            recipient: msg.sender,
            amount0Max: type(uint128).max,
            amount1Max: type(uint128).max
        })
    );
```

Recommendation: If during the deposit any fees are collected, send them to the owner of the position.

Mellow: Fixed in commit 52c0aaf0.

Cantina Managed: Fixed. Upon deposit, fees are now sent to the position owner.

3.3.5 Denial of service attack on `LpWrapper`'s `deposit` and `withdraw` functions

Severity: Medium Risk

Context: Counter.sol#L26-L29 Core.sol#L98

Description: The `Core` contract allows any address to be set as the `CallbackParams.counter` address of a managed position. The LpWrapper also has its own `CallbackParams.counter` set in its managed position.

Combination of the above stated mechanisms can be used to DoS the `deposit` and `withdraw` functions of `LpWrapper`.

Scenario:

- An attacker brings an NFT with insignificant (but >0) liquidity and deposits it in Core.
- Attacker updates the `gauge` and `counter` addresses of his position such that:
  - `gauge` returns a malicious `rewardToken` which returns a huge `uint256` value as Core contract balance.
  - `counter` is set to the `Counter` contract which is used by an `LpWrapper` instance.
- Attacker performs an `emptyRebalance` of his position. Here the `value` state of `LpWrapper's Counter` becomes `uint256.max`.
- Now no more rewards can be counted in `LpWrapper's Counter` as the `Counter.add` call will always reverts due to overflow.
- As on every `LpWrapper`'s `deposit` and `withdraw` call the flow goes like this `LpWrapper.deposit()` → `Core.withdraw()` → `VeloAmmModule.beforeRebalance()` → `Counter.add()`.
- All `deposit` and `withdraw` transactions will revert. Resulting in DoS of `LpWrapper`.

Recommendation: Consider dropping the use of `Counter` contract as there is no on-chain use of this contract, rewards sent from Core to Farm can be tracked off-chain. Or consider removing the ability to choose or change the `Counter` address of a managed position.

Mellow: Fixed 736eef90.

Cantina Managed: The `Counter.add` function now validates the `token` and `farm` addresses. These validations prevent the above mentioned attack.

3.3.6 Unsafe max `deadline` provided

Severity: Medium Risk

Context: VeloDepositWithdrawModule.sol#L36, VeloDepositWithdrawModule.sol#L59, VeloDeployFactory.sol#L155

Description: A maximum deadline is provided for execution of multiple `NonfungiblePositionManager` functions: `increaseLiquidity`, `decreaseLiquidity` and `mint`:

```
deadline: type(uint256).max,
```

Note also that the following format provided in `VeloDeployFactory._mint` is also effectively a max deadline because it marks the deadline as the current timestamp at the time of execution, which effectively causes the deadline check to validate that `block.timestamp <= block.timestamp`, which is always true:

```
deadline: block.timestamp,
```

Use of a maximum deadline is unsafe as it may result in the transaction to sit in the mempool for a very long time, until the conditions in which we're modifying our liquidity position have become undesirable, potentially even as a result of an MEV builder processing the transactions alongside some other actions to intentionally cause a loss of funds.

Recommendation: Rather than using a maximum deadline, consider allowing the caller to provide the `deadline` as a parameter.

Mellow: Fixed in 09171ab5.

Cantina Managed: Issue has been fixed by allowing user to provide `deadline` parameter which is validated in the outer call context.

3.3.7 Lack of consideration of time in `ensureNoMEV` leads to unexpected reverts

Severity: Medium Risk

Context: VeloOracle.sol#L11-L53

Description: `VeloOracle.ensureNoMEV` works by looking back at a given amount of `observations` and reverting if the `tick` delta between `observations` exceeds a `maxAllowedDelta`. The `observations` are retrieved from the given `CLPool` where they are written any time an in-range liquidity position is modified, or a swap occurs, updating the most recent observation if it was within 15 seconds. This logic is used to ensure that the price of the pool has not *recently* changed beyond a certain amount.

The problem with `ensureNoMEV`'s logic here is that it implicitly assumes that the `observations` that are being considered are recent, however if there have not been any swaps or in-range liquidity modifications recently, then there will not be recent `observations`. Instead, we may be looking further back in time than intended.

Consider for example a circumstance where there's a large swap that exceeds the `maxAllowedDelta`, then there is no activity for a while. Even though the price has not recently changed, the function will revert since it's looking at the most recent observations, regardless of when they actually occurred.

The result of this is that rebalances can be DoS'd until sufficient `observations` are written such that `ensureNoMEV` no longer reverts. This can lead to extended periods of time in which the position is out of range and thus not earning liquidity fees.

Recommendation: Consider modifying the `ensureNoMEV` logic to consider time between `observations` instead of just a `lookback` amount of `observations`, e.g. if there are no `observations` exceeding the `maxAllowedDelta` in `x` seconds, return.

Mellow: Fixed in commit f839c6c8.

Cantina Managed: Issue has been fixed by incorporating a `maxAge` to determine whether `observations` should be evaluated. Fix also persist and is present in `736eef90ecfa896b12b5f193e68bf95030eb475e`.

3.3.8 Unexpected ETH transfer DoS

Severity: Medium Risk

Context: VeloDepositWithdrawModule.sol#L29-L38, VeloDeployFactory.sol#L143-L158

Description: The `NonfungiblePositionManager` has a few functions which include execution of the `refundETH` function. In `VeloDepositWithdrawModule.deposit` and `VeloDeployFactory.createStrategy`, we call `NonfungiblePositionManager.increaseLiquidity` and `NonfungiblePositionManager.mint`, respectively. Both of these functions execute `refundETH`.

`refundETH` works by simply transferring any remaining ETH balance in the contract to the `msg.sender`:

```
function refundETH() public payable override nonReentrant {
    if (address(this).balance > 0) TransferHelper.safeTransferETH(msg.sender, address(this).balance);
}
```

The problem with this is that neither `VeloDepositWithdrawModule` nor `VeloDeployFactory` are capable of receiving ETH since they don't have a `receive` or `fallback payable` functions. See the Solidity documentation:

> "When Ether is sent directly to a contract (without a function call, i.e. sender uses send or transfer) but the receiving contract does not define a receive Ether function or a payable fallback function, an exception will be thrown, sending back the Ether".

As a result, any time the `NonfungiblePositionManager` contract has a non-zero ETH balance, `VeloDepositWithdrawModule.deposit` and `VeloDeployFactory.createStrategy` will unexpectedly revert. This can happen by chance or it can be intentionally exploited by an attacker by frontrunning `deposit` and `createStrategy` to transfer ETH to the `NonfungiblePositionManager` using `SELFDESTRUCT/SENDALL`, or even by using one of the `payable` functions which don't refund ETH.

Recommendation: To resolve this issue, it's necessary to add a `receive` or `fallback payable` function to both the `LpWrapper` and `VeloDeployFactory` contracts. Additionally, it's important to consider a mechanism for handling received ETH afterwards, perhaps by transferring the ETH directly to a pre-defined address capable of receiving ETH.

Mellow: Fixed in PR 50.

Cantina: Fixed by including a `receive` function that wraps the ETH and transfers to the `tx.origin`.

3.4 Low Risk

3.4.1 Multi-block MEV may still be possible with `ensureNoMEV`

Severity: Low Risk

Context: VeloOracle.sol#L43-L50

Description: The `VeloOracle.ensureNoMEV` function takes a `lookback` amount of `observations` to look at and reverts if the `tick` change between individual `observations` exceeds the `maxAllowedDelta`.

```
for (uint16 i = 1; i <= lookback; i++) {
    uint256 index = (observationCardinality + observationIndex - i) %
        observationCardinality;
    (uint32 timestamp, int56 tickCumulative, , ) = ICLPool(poolAddress)
        .observations(index);
    if (timestamp == 0) revert NotEnoughObservations();
    int24 tick = int24(
        (nextCumulativeTick - tickCumulative) /
            int56(uint56(nextTimestamp - timestamp))
    );
    (nextTimestamp, nextCumulativeTick) = (timestamp, tickCumulative);
    int24 delta = nextTick - tick;
    if (delta > maxAllowedDelta || delta < -maxAllowedDelta)
        revert PriceManipulationDetected();
    nextTick = tick;
}
```

The concern with this logic is that the price change over a `lookback` amount of `observations` may be significant while still being valid between individual observations according to the `maxAllowedDelta`. This can be manipulated by an attacker watching the mempool if they can predict a minimum amount of blocks that it would take for the rebalancing transaction to be accepted. In this time, the attacker could strategically frontrun the transaction over multiple observation periods such that we go up to but not exceed the `maxAllowedDelta` for any given observation period.

An attacker could predict the minimum amount of blocks a transaction would take to be processed simply by considering the gas price of each transaction in the mempool, taking the sum of the gas limits of all transactions with a higher gas price and dividing that sum by the block gas limit.

Marking this as low severity since it has been acknowledged as being within the intended design of the protocol, though this particular attack vector may not be considered.

Recommendation: Consider validating against a total max delta of the sum of all observation deltas. Alternatively, consider adding documentation to indicate that this kind of attack may be possible.

Mellow: Acknowledged.

Cantina Managed: Acknowledged.

3.4.2 The `params` provided to `VeloDeployFactory.updateStrategyParams` is not validated

Severity: Low Risk

Context: VeloDeployFactory.sol#L46-L52

Description: Checks similar to `validateStrategyParams` are missing.

Recommendation: Make sure to add the same type of validation checks for this endpoint.

Mellow: The following changes have been introduced:

1. `VeloDeployFactory.updateStrategyParams` has been removed.
2. The created `strategyParams`:

   ```
   IPulseStrategyModule.StrategyParams memory strategyParams = IPulseStrategyModule.StrategyParams({
       tickNeighborhood: params.tickNeighborhood,
       tickSpacing: int24(position.property),
       strategyType: params.strategyType,
       width: position.tickUpper - position.tickLower
   });
   ```

   is later validated when `core.deposit(depositParams)` is called

   ```
   lpWrapper.initialize(core.deposit(depositParams), position.liquidity);
   ```

   Sidenote, `params.securityParams` is used first:

   ```
   core.oracle().ensureNoMEV(address(pool), params.securityParams);
   ```

   Then later validated in `core.deposit(depositParams)`:

   ```
   lpWrapper.initialize(core.deposit(depositParams), position.liquidity);
   ```

Fixed in commit 736eef90.

Cantina: Verified.

3.4.3 `LpWrapper`'s `initialize` can be called by anyone to set and fix most of the relevant parameters

Severity: Low Risk

Context: LpWrapper.sol#L51-L61

Description: Anyone can donate/deposit their position in `Core` to `LpWrapper` and then call `initialize` with that position and their desired `initialTotalSupply`. Thus fix the pool ($T_0, T_1, \Delta i$) associated to this contract. As after initialisation this pool cannot be changed by the other endpoints.

Moreover they can choose the number of concentrated liquidity ranges and the initial tick indices and their associated liquidity. They can also set the callback, strategy and security parameters until an `LpWrapper` admin calls the `setPositionParams`.

> In `VeloDeployFactoryHelper` the `createLpWrapper` does not call the `constructor` and `initialize` atomically.
>
> But in `VeloDeployFactory.createStrategy` the calls to `constructor` and `initialize` happen in the same frame.

Recommendation: Either a comment should be added for those who want to deploy this contract without using `VeloDeployFactory.createStrategy` as a warning. Or restrict `initialize` to be only callable by some specific addresses/entities.

Mellow: Acknowledged.

Cantina Managed: Acknowledged.

3.4.4 Checks missing for the derived contracts addresses in `LpWrapper.constructor`

Severity: Low Risk

Context: LpWrapper.sol#L46-L48

Description: In the `LpWrapper.constructor` one derives and stores the following addresses:

```
positionManager = ammModule.positionManager();
ammDepositWithdrawModule = ammDepositWithdrawModule_;
```

`ammDepositWithdrawModule` also has its own immutable `positionManager` address.

We need to make sure that the derived `positionManager` matches with the one stored in `ammDepositWithdrawModule`.

Recommendation: Add a check in the `constructor` to make sure these `2` addresses match:

```
ammDepositWithdrawModule_.positionManager() == ammModule.positionManager()
```

Mellow: Acknowledged.

Cantina Managed: Acknowledged.

3.4.5 Unusability of `LpWrapper`'s `deposit` and `withdraw` functions in case the contract is initialized with zero `initialTotalSupply`

Severity: Low Risk

Context: LpWrapper.sol#L60 Core.sol#L123

Description: The `LpWrapper` is assumed to be a modular contract. The `LpWrapper` is initialized with an `initialTotalSupply` value. This is the amount of LpWrapper ERC20 tokens which gets locked in the contract forever. This prevents users from withdrawing all deposited tokens from the LpWrapper and empty the contract.

However in case the `LpWrapper` gets initialized with an `initialTotalSupply` of `0` then all users will be able to withdraw all funds and empty the pool. But the withdrawal for the last withdrawer will always fail because his withdrawal will make the `position.liquidity` of `ammPositionIds` to be `0` and such NFT cannot be deposited into `Core` (due to Core.sol#L123).

Hence the `LpWrapper.withdraw` transaction of last withdrawer will always fail. Moreover, due to these statements the last withdrawer cannot withdraw partial amounts. Mitigation would be to deposit some dust amount into LpWrapper and keep it locked forever.

Under current protocol setup, it is assumed that `LpWrapper` will be deployed via `VeloDeployFactory`. The `0` `initialTotalSupply` situation is not possible in that way of deployment.

Recommendation: In the `LpWrapper.initialize` function make sure that `initialTotalSupply` is not `0`, preferably it should be above a certain threshold.

Mellow: Fixed in commit 736eef90.

Cantina Managed: Issue has been fixed by adding a check in `LpWrapper.initialize` which ensures that `initialTotalSupply` is not zero.

3.4.6 Unusability of `LpWrapper`'s `deposit` and `withdraw` functions in case a position's liquidity becomes 0

Severity: Low Risk

Context: LpWrapper.sol#L95 Core.sol#L123

Description: The `LpWrapper` is assumed to be a modular contract and should be able to handle multiple `ammPositionIds`. During deposits, it proportionately allocates the input token amounts to existing liquidities of `ammPositionIds` NFTs. However it doesn't add any amount to NFTs whose liquidity is already 0. After adding the proportionate amounts to all non-zero liquidities it tries to deposit all `ammPositionIds` into the `Core` contract.

But the `Core.deposit` has this statement `if (position_.liquidity == 0) revert InvalidParams();`, so Core does not accept any NFT with zero liquidity.

In case the liquidity of an existing LpWrapper position's `ammPositionIds` becomes 0 (due to rounding) then the `LpWrapper`'s `deposit` and `withdraw` functions will start getting reverted on every call making the contract unusable.

Recommendation: Consider ejecting the `ammPositionIds` from the array in case its liquidity goes to 0 (& transfer NFT to admin). Or at deployment make sure that liquidities of all `ammPositionIds` are above a certain threshold so their liquidity going to 0 becomes less likely.

Mellow: Fixed in commit 736eef90.

Cantina Managed: Issue has been fixed as `Core.deposit` now only reverts when liquidity of all `ammPositionIds` positions is zero.

3.4.7 `LpWrapper` deposits and withdraws may be temporarily bricked in a certain edge case.

Severity: Low Risk

Context: LpWrapper.sol#L255

Description: `LpWrapper` is expected to be able to work with multiple `ammPositionIds` at the same time. Due to strategy or rounding down, it can happen so that one of the positions gets to 0 liquidity. Since upon every `withdraw/deposit`, all positions are withdrawn and deposited back in `Core.sol` and `Core#deposit` requires all positions to have non-zero liquidity, this would force all `LpWrapper` deposits and withdraws to be bricked.

Since users will not be able to directly donate liquidity to the position, only way to unbrick it would be for a rebalancer to call `rebalance` and donate liquidity within the `RebalanceCallback` contract.

Recommendation: Do not revert on 0 liquidity position deposits.

Mellow: Fixed in commit 09171ab5.

Cantina Managed: Fixed in commit 09171ab5.

3.4.8 Always centering the position if `width` is changed may lead to unexpected behaviour.

Severity: Low Risk

Context: PulseStrategyModule.sol#L92

Description: If a user has Lazy Ascending or Lazy Descending strategy, it would aim to always rebalance the position in a single-side liquidity way. If the user has just rebalanced their position and they wish to change the position's width, it will not be done according to the user's strategy, but rather it will always be centered.

```
        if (params.width != tickUpper - tickLower)
            return _centeredPosition(tick, params.width, params.tickSpacing);
```

Recommendation: Mellow: Acknowledged.

Cantina Managed: Acknowledged.

3.4.9 `rebalance` might unnecessarily revert when rebalancing multiple positions

Severity: Low Risk

Context: VeloOracle.sol#L29

Description: When rebalancing multiple positions, for each position `ensureNoMEV` is called to check against pool price manipulations, according to each position's own `securityParams`. When multiple positions are being rebalanced, if only one of them puts unreasonable `securityParams`, it will cause the whole transaction to revert and all rebalances to fail.

Recommendation: In case MEV is detected, consider continuing the rebalancing for other positions.

Mellow: Acknowledged.

Cantina Managed: Acknowledged.

3.4.10 LpWrapper contracts can be deployed without oracle security parameters

Severity: Low Risk

Context: VeloDeployFactory.sol#L60

Description: The Mellow protocol utilizes oracle `SecurityParams` to prevent price manipulation attacks. Currently it is possible for the admin the set empty `SecurityParams` for a `tickSpacing` using the `updateDepositParams` function. Hence it is also possible for the `OPERATOR` to deploy an `LpWrapper` for a `CLPool` with empty `SecurityParams`. That kind of `LpWrapper` will be susceptible to price manipulation attacks. Since only one `LpWrapper` can be deployed per `CLPool` and `LpWrapper` manages pooled funds of users, that scenario won't be ideal.

Recommendation: Consider enforcing that the length of `DepositParams.SecurityParams` is not zero:

```
  function updateDepositParams(
      int24 tickSpacing,
      ICore.DepositParams memory params
  ) external {
      _requireAdmin();
+     if (params.securityParams.length == 0) revert InvalidParams();
      _tickSpacingToDepositParams[tickSpacing] = params;
  }
```

Mellow: Fixed 736eef90.

Cantina Managed: The `VeloDeployFactory.createStrategy` now validates that `params.securityParams.length` is not zero.

3.4.11 Unoptimal use of predetermined `initialLiquidity` for `LpWrapper` creation

Severity: Low Risk

Context: VeloDeployFactory.sol#L132

Description: The `VeloDeployFactory._mint` function uses a prefixed and same amount of `initialLiquidity` for all ERC20 tokens (with different decimals) which is not optimal. The `initialLiquidity` which works for WETH-OP pool will not work for USDC-USDT pool, as the computed `amount0` and `amount1` for the latter one will be significantly larger.

This initial liquidity gets locked in the `LpWrapper` contract forever so if the `amount0` and `amount1` are significantly large then it will be difficult for strategy creator to lock that much amount of funds.

Admin will often need to adjust the `initialLiquidity` param every time a strategy needs to be deployed for a token pair.

Recommendation: Since `VeloDeployFactory.createStrategy` is an access restricted function consider taking the `initialLiquidity` as input for every strategy deployment.

Mellow: Fixed in commit 736eef90.

Cantina Managed: Issue has been fixed as the mechanism to take individual `token0` and `token1` amounts from the caller of `VeloDeployFactory.createStrategy` has been removed. Now the

VeloDeployFactory.createStrategy function takes an amm position NFT from the caller. Hence creating the initial position with individual token amounts is not needed.

3.4.12 During strategy creation the rewardToken should be fetched dynamically

Severity: Low Risk

Context: VeloDeployFactory.sol#L245

Description: The VeloDeployFactory.Storage.MutableParams.rewardsToken stores the reward token address which is required at the time of StakingRewards contract deployment. This is the address of ERC20 tokens which are received from Gauge as rewards.

Technically the reward token of a Gauge can be different for every Gauge instance (see CLGaugeFactory.sol#L49).

So the reward token address should be dynamically read by calling ICLGauge(pool.gauge()).rewardToken() instead of storing it as a fixed parameter. This reduces chance of setting an incorrect token.

Recommendation: Consider reading the reward token address from Gauge dynamically.

```diff
  address farm = s.immutableParams.helper.createStakingRewards(
      s.mutableParams.farmOwner,
      s.mutableParams.farmOperator,
-     s.mutableParams.rewardsToken,
+     ICLGauge(pool.gauge()).rewardToken(),
      address(lpWrapper)
  );
```

Mellow: Fixed 736eef90.

Cantina Managed: Issue has been fixed as the rewardToken address is now being read from Gauge dynamically.

3.4.13 The LpWrapper::OPERATOR role is not revoked in VeloDeployFactoryHelper.createLpWrapper function

Severity: Low Risk

Context: DefaultAccessControl.sol#L24 VeloDeployFactoryHelper.sol#L26

Description: The LpWrapper contract inherits DefaultAccessControl. When the VeloDeployFactoryHelper deploys a new LpWrapper the OPERATOR is granted automatically to the VeloDeployFactoryHelper. This role is not revoked after the deployment of LpWrapper.

Recommendation: Consider revoking the LpWrapper::OPERATOR role in VeloDeployFactoryHelper.createLpWrapper function:

```diff
  wrapper.grantRole(wrapper.OPERATOR(), operator);
+ wrapper.revokeRole(wrapper.OPERATOR(), address(this));
  wrapper.revokeRole(wrapper.ADMIN_DELEGATE_ROLE(), address(this));
```

Mellow: Fixed in commit 736eef90.

Cantina Managed: Issue has been fixed as the OPERATOR role is now renounced in VeloDeployFactoryHelper.createLpWrapper function. .

3.4.14 Withdrawal flow of users will get broken if their position's `CallbackParams` are set as null

Severity: Low Risk

Context: Core.sol#L98 VeloAmmModule.sol#L139

Description: In case a user creates a position in Core with non-null `CallbackParams` and then sets those params to null value then his withdraw transaction cannot be processed. The opposite situation is also possible when a position gets created will null callback params which then gets updated to a non-null value.

Scenario:

- User deposits his liquidity NFTs into Core with address `0x1234...` as the `CallbackParams.gauge` address. The deposited NFT gets staked into the gauge.
- User sets his position's `CallbackParams` as null (`callbackParams.length = 0`) using the `setPosition-Params` function.
- Now when user calls `Core.withdraw` the Core contract will try to return the NFT to user without pulling it from gauge, which will result in a revert.

Recommendation: Consider not allowing new callback params to be null if they were not null previously (and vice versa). Also if there is no explicit need to change `CallbackParams` of a position then remove this feature from the contract.

Mellow: Fixed commit 736eef90.

Cantina Managed: Issue has been fixed as the `CallbackParams` cannot be set as null now. Also the `CallbackParams.gauge` address is being validated.

3.4.15 Invalid `ManagedPositionInfo`s can be created in `Core`

Severity: Low Risk

Context: Core.sol#L117

Description: The deposit function iterates over the `DepositParams.ammPositionIds` array to pull liquidity NFT tokens from caller. But in case this `ammPositionIds` array is provided as an empty array then no NFTs will be pulled from caller but still an `ManagedPositionInfo` struct will be created and pushed into `_positions` array.

Hence malicious users can create infinite invalid and unusable `ManagedPositionInfo` positions in Core contract.

Recommendation: Consider validating that the length of `ammPositionIds` is not zero.

```
if (params.ammPositionIds.length == 0) revert InvalidParams();
```

Mellow: Fixed in commit 736eef90.

Cantina Managed: Issue has been fixed as the `Core.deposit` function now implements a `hasLiquidity` boolean flag. In case the length of `DepositParams.ammPositionIds` is `0` then the function will revert.

3.4.16 Lack of authorization for functions intended to be `delegatecall`ed

Severity: Low Risk

Context: VeloAmmModule.sol#L189, VeloAmmModule.sol#L134, VeloDepositWithdrawModule.sol#L17, VeloDepositWithdrawModule.sol#L48

Description: The protocol often uses module contracts with some functions which are intended to only be executed using `delegatecall` from other contracts in the system. This can lead to unexpected effects when the functions are called directly.

`VeloAmmModule.transferFrom` can be called by anyone to transfer any `positionManager` tokens arbitrarily. In case anyone approves this contract instead of the `Core` contract which is delegatecalling it, their tokens can be stolen.

```
function transferFrom(
    address from,
    address to,
    uint256 tokenId
) external virtual override {
    INonfungiblePositionManager(positionManager).transferFrom(
        from,
        to,
        tokenId
    );
}
```

Similarly, in `VeloAmmModule.beforeRebalance`, users can skim any ERC20 tokens which end up in this contract since they can arbitrarily control the parameters. Furthermore, users who call `VeloDepositWithdrawModule.deposit` directly can have their position withdrawn by anyone in `VeloDepositWithdrawModule.withdraw`.

Recommendation: Ensure these functions can only be delegatecalled by enforcing that `address(this)` is not the contract address, e.g. by using a modifier like the following:

```
// NOTE: This code is untested
address constant THIS = address(this);

modifier onlyDelegateCall() {
    if (address(this) == THIS) revert NOT_DELEGATE_CALL();
}
```

Mellow: Acknowledged.

Cantina Managed: Acknowledged.

3.4.17 `rewardsToken` may cause rounding issues if not 18 decimals

Severity: Low Risk

Context: VeloDeployFactory.sol#L64-L67

Description: `ICLGauge.notifyRewardWithoutClaim` documents the following:

```
/// Assumes gauge reward tokens is 18 decimals.
/// If not 18 decimals, rewardRate may have rounding issues.
```

The reward token used with the `CLGauge` contract is `_contractStorage().mutableParams.rewardsToken`, which is set in `VeloDeployFactory.updateMutableParams`:

```
function updateMutableParams(MutableParams memory params) external {
    _requireAdmin();
    _contractStorage().mutableParams = params;
}
```

Recommendation: To avoid rounding issues with non-18-decimal tokens, it's recommended that logic is added in `VeloDeployFactory.updateMutableParams` to validate that `params.rewardsToken` has 18 decimals.

Mellow: Fixed in 8c007d5e.

Cantina Managed: Issue is fixed by using `gauge.rewardToken` instead of arbitrary token:

```
address rewardToken = ICLGauge(gauge).rewardToken(); // <--- here ---
poolAddresses.synthetixFarm = immutableParams
    .helper
    .createStakingRewards(
        mutableParams.farmOwner,
        mutableParams.farmOperator,
        rewardToken,                              // <--- here ---
        address(lpWrapper)
    );
depositParams.callbackParams = abi.encode(
    IVeloAmmModule.CallbackParams({
        farm: poolAddresses.synthetixFarm,
        gauge: address(gauge),
        counter: address(
            new Counter(
                mutableParams.farmOperator,
                address(core),
                rewardToken,                      // <--- here ---
                poolAddresses.synthetixFarm
            )
        )
    })
);
```

and `rewardsToken` has been removed from:

```
struct MutableParams {
    address lpWrapperAdmin; // Admin address for the LP wrapper
    address lpWrapperManager; // Manager address for the LP wrapper
    address farmOwner; // Owner address for the farm
    address farmOperator; // Operator address for the farm (compounder)
    uint256 minInitialLiquidity; // Minimum initial liquidity for the LP wrapper
}
```

Fix persist in `736eef90ecfa896b12b5f193e68bf95030eb475e`.

3.4.18 `rebalance` may revert for positions with multiple `ammPositionIds`

Severity: Low Risk

Context: PulseStrategyModule.sol#L42-L44

Description: It's possible to create a managed position in `Core` with an arbitrary number of `ammPositionIds`, but `PulseStrategyModule` only supports one:

```
if (info.ammPositionIds.length != 1) {
    revert InvalidLength();
}
```

Presumably, the intention here is to be able to use the same `Core` contract code on different deployments with different strategy modules (note that the `strategyModule` is immutable). However, the problem is that this allows for users to `deposit` multiple `ammPositionIds` on the deployment using `PulseStrategyModule` even though `rebalance` will always revert.

Recommendation: If the intention is to use the same `Core` contract code on different deployments with different strategy modules, include a parameter in `strategyModule.validateStrategyParams` to pass `ammPositionsIds.length`, where in `PulseStrategyModule.validateStrategyParams`, we revert if `length != 1`.

Mellow: Acknowledged.

Cantina Managed: Acknowledged.

3.4.19 Anyone can skim ERC20 tokens from the `Core` contract

Severity: Low Risk

Context: VeloAmmModule.sol#L150-L172

Description: In `VeloAmmModule.beforeRebalance`, which is `delegatecalled` by `Core` to retrieve a `tokenId` from the `gauge` contract, we retrieve rewards from the `gauge` and transfer those rewards, excluding a protocol fee, to the `farm` contract where stakers can later receive the rewards.

```
ICLGauge(callbackParams_.gauge).getReward(tokenId);
address token = ICLGauge(callbackParams_.gauge).rewardToken();
uint256 balance = IERC20(token).balanceOf(address(this));
if (balance > 0) {
    uint256 protocolReward = FullMath.mulDiv(
        protocolParams_.feeD9,
        balance,
        D9
    );

    if (protocolReward > 0) {
        IERC20(token).safeTransfer(
            protocolParams_.treasury,
            protocolReward
        );
    }

    balance -= protocolReward;
    if (balance > 0) {
        IERC20(token).safeTransfer(callbackParams_.farm, balance);
        ICounter(callbackParams_.counter).add(balance);
    }
}
```

The problem is that the `callbackParams`, which contain the `gauge` and `farm` contracts, are only validated to not be `address(0)`, thus users can arbitrarily set these addresses:

```
if (params_.farm == address(0)) revert AddressZero();
if (params_.gauge == address(0)) revert AddressZero();
if (params_.counter == address(0)) revert AddressZero();
```

An attack may proceed as follows:

- Attacker sets `gauge` to be a contract they control which returns any token they decide when `rewardToken()` is called.
- Attacker sets `farm` as an address they control to receive funds.
- Attacker makes a deposit with above `callbackParams` and immediately withdraws the position, triggering the `beforeRebalance` hook.

The result of this is that any ERC20 tokens in the `Core` contract can be withdrawn by the attacker.

Recommendation: Include an allowlist of contracts to be used for each of the `callbackParams` in `VeloAmmModule.validateCallbackParams`.

Mellow: Fixed in commit 06b459b2. `validateCallbackParams` now performs the following check:

```
function validateCallbackParams(bytes memory params) external view {
    // ...
    ICLPool pool = ICLGauge(params_.gauge).pool();
    if (!factory.isPair(address(pool))) revert InvalidGauge();
    if (pool.gauge() != params_.gauge) revert InvalidGauge();
}
```

So before `_positions[id]` gets storied in the storage, it is verified that the `gauge` and its associated `pool` are stemming from the the registered `factory` contract in `VeloAmmModule`.

Cantina Managed: Issue is fixed by enforcing that a valid `gauge` is used. Fix persist in commit 736eef90.

3.5 Gas Optimization

3.5.1 `0` can be passed as lower and upper ticks when `strategyModule.calculateTarget` in `VeloDeployFactory._mint`

Severity: Gas Optimization

Context: VeloDeployFactory.sol#L118-L119

Description: In this context we call `strategyModule.calculateTarget` with the following parameters:

```
( /*...*/ ) = strategyModule.calculateTarget(
    // ...
    type(int24).min, // tickLower
    type(int24).min, // tickUpper
    // ...
);
```

We could have also passed `0` here. Since `strategyParams.intervalWidth` should be non-zero. And in `PulseStrategyModule.calculateTarget --> _calculatePosition` we enter into the following `if` block:

```
if (params.width != tickUpper - tickLower) // <--- tickUpper and tickLower are equal either in the
↪  `type(int24).min` case or even when passing `0`.
    return _centeredPosition(tick, params.width, params.tickSpacing);
```

and as long as `tickUpper == tickLower` the above statement is equivalent to `params.width != 0`. The only other concern is that when “ are calculated we would want to not fall into the following `if` block:

```
if (targetTickLower == tickLower && targetTickUpper == tickUpper)
   return (false, target);
```

or equivalently:

```
if (targetTickLower == 0 && targetTickUpper == 0)
   return (false, target);
```

But this conditional statement cannot be true, since for centred positions the difference between the target upper and lower ticks are `params.width` which is non-zero.

Recommendation: To lower the gas costs we can provide `0` as both the upper and lower ticks params provided to `strategyModule.calculateTarget`:

> Since the above change depends on many different indirect invariants/conditions, if any of those change in the future it might break the assumption where we can replace `type(int24).min` with `0`.

Mellow: `target` in `createStrategy` is calculated as:

```
(
    bool isRebalanceRequired,
    ICore.TargetPositionInfo memory target
) = immutableParams.strategyModule.calculateTarget(
    tick,
    0,
    0,
    strategyParams
);
```

using the recommendation.

Cantina: Fixed in commit 736eef90.

3.5.2 Redundant allowance and balance check before transfer

Severity: Gas Optimization

Context: VeloDeployFactory.sol#L74-L92

Description: In `VeloDeployFactory._prepareToken`, prior to transferring tokens from `msg.sender`, we check whether the `allowance` to `address(this)` and `balanceOf` the sender are sufficient to execute the transfer, reverting if the amounts are insufficient:

```
uint256 allowance = IERC20(token).allowance(msg.sender, address(this));
uint256 userBalance = IERC20(token).balanceOf(msg.sender);
if (allowance < amount || userBalance < amount)
    revert(
        string(
            abi.encodePacked(
                "Invalid ",
                IERC20Metadata(token).symbol(),
                " allowance or balance. Required: ",
                Strings.toString(amount),
                "; User balance: ",
                Strings.toString(userBalance),
                "; User allowance: ",
                Strings.toString(allowance),
                "."
            )
        )
    );
IERC20(token).safeTransferFrom(msg.sender, address(this), amount);
```

These balance and allowance checks are redundant because they're checked again in `transferFrom` regardless.

Note: *If the intention here is to have a readable revert message and the value of that outweighs the additional gas cost, then this is an acceptable design decision.*

Recommendation: Simply execute the `safeTransferFrom` without the redundant checks:

```
- uint256 allowance = IERC20(token).allowance(msg.sender, address(this));
- uint256 userBalance = IERC20(token).balanceOf(msg.sender);
- if (allowance < amount || userBalance < amount)
-     revert(
-         string(
-             abi.encodePacked(
-                 "Invalid ",
-                 IERC20Metadata(token).symbol(),
-                 " allowance or balance. Required: ",
-                 Strings.toString(amount),
-                 "; User balance: ",
-                 Strings.toString(userBalance),
-                 "; User allowance: ",
-                 Strings.toString(allowance),
-                 "."
-             )
-         )
-     );
  IERC20(token).safeTransferFrom(msg.sender, address(this), amount);
```

Mellow: Fixed by changing strategy creation mechanism.

Cantina Managed: Issue is no longer present due to aforementioned change to strategy creation.

3.5.3 Redundant zero value state var initialization

Severity: Gas Optimization

Context: Counter.sol#L8

Description: In `Counter`, we initialize `value` as 0:

```
uint256 public value = 0;
```

However, since the default value for `uint256` variables is already 0, this stores the same value which is already present. A cold `SSTORE` from a zero value to a zero value like this costs 2200 gas.

Recommendation: Remove the redundant re-initialization:

```
- uint256 public value = 0;
+ uint256 public value;
```

Mellow: Acknowledged.

Cantina Managed: Acknowledged.

3.5.4 Off-chain counter mechanism

Severity: Gas Optimization

Context: VeloAmmModule.sol#L170

Description: In `VeloAmmModule.beforeRebalance`, we increment the `Counter` contract state by the `balance` amount of tokens being transferred to the `farm` contract by calling `Counter.add`:

```
if (balance > 0) {
    IERC20(token).safeTransfer(callbackParams_.farm, balance);
    ICounter(callbackParams_.counter).add(balance);
}
```

This requires both a cold `CALL` and a cold `SSTORE`, with the `CALL` costing 2600 gas and the `SSTORE` costing 5000 gas (or 20000 gas the first time `add` is called).

Recommendation: If the amount tracked by `Counter` is not directly needed on-chain, a significant amount of gas could be saved by emitting an event instead and indexing the amount off-chain by listening for the event.

Mellow: Acknowledged.

Cantina Managed: Acknowledged.

3.6 Informational

3.6.1 Lack of testing

Severity: Informational

Context: test/

Description: Throughout the codebase, there are some areas in which the testing is insufficient to provide a high degree of confidence in the logic correctness. Some particular areas of concern:

- `Core.rebalance`.
- `VeloDeployFactory` (lack of unit tests).

Additionally, it would be good to see a wider variety of tests that validate not just that the logic succeeds in base cases under normal conditions, but also: negative, fuzzing, and invariant tests.

Recommendation: Add testing throughout the codebase with a focus on the above listed areas and methods.

Mellow: Acknowledged.

Cantina Managed: Acknowledged.

3.6.2 Some of the `_tickSpacingToDepositParams` fields are unused

Severity: Informational

Context: VeloDeployFactory.sol#L15

Description: `_tickSpacingToDepositParams` has the following value form:

```
struct DepositParams {
    uint256[] ammPositionIds;
    address owner;
    uint16 slippageD4;
    bytes callbackParams;
    bytes strategyParams;
    bytes securityParams;
}
```

Recommendation: Even though only the `slippageD4` and `securityParams` values are actually used in `createStrategy` and the other fields are calculated just-in-time during the flow of `createStrategy` and any other stored values are ignored for those fields.

Perhaps `_tickSpacingToDepositParams` can be renamed to `_tickSpacingToSlippageAndSecurityParams` with its value of the following form:

```
struct SlippageAndSecurityParams {
    uint16 slippageD4;
    bytes securityParams;
}
```

Mellow: `_tickSpacingToDepositParams` storage parameter has been removed and the input variable to `createStrategy` has been changed to `DeployParams calldata params`:

```
struct DeployParams {
    int24 tickNeighborhood;
    uint32 slippageD9;
    uint256 tokenId;
    bytes securityParams;
    IPulseStrategyModule.StrategyType strategyType;
}
```

and so the flow has also been changed to accommodate only one `position` instead of having a general field of `ammPositionIds`.

Cantina: Fixed in commit 736eef90.

3.6.3 Unreachable `revert` statement

Severity: Informational

Context: VeloDeployFactory.sol#L128

Description: In this context we have:

```
(bool isRebalanceRequired, ...) = strategyModule.calculateTarget(
    tick,
    type(int24).min,
    type(int24).min,
    ...
);

if (!isRebalanceRequired) revert InvalidState(); // <--- this should be unreachable
```

Since the used strategy is `IPulseStrategyModule.StrategyType.Original` and provided lower and upper ticks are equal. But the target ones returned should have a difference of non-zero `width`.

Recommendation: Perhaps above can be documented and different test cases can be added in the test suite. The `if` statement can still stay where it is in case of future implementations as a general `IPulseStrategyModule` might return different parameters given the same specific input parameters.

Mellow: The `require` statement has been transformed into:

```
assert(isRebalanceRequired);
```

Cantina: Fixed in commit 736eef90.

3.6.4 Some `LpWrapper` invariants

Severity: Informational

Context: LpWrapper.sol#L184, LpWrapper.sol#L258

Description: The `info.owner` will be just `address(this)` in this context. as during the lifetime of the `LpWrapper` contract, the `owner` parameter stays the same as the contract.

In fact during the lifetime of `LpWrapper` the followings stay the same for any position id of the `Core` contract it consumes:

- `property` (cached Δi `tickSpacing` for the cases used in the current codebase).
- `pool` (which means the $T_0, T_1, \Delta i$ stay the same).
- `owner` (or `address(this)`).

Recommendation: Perhaps these invariants can be tested in the test suite. Also as an optimisation and/or hardcoded invariant the `info.owner` can be replaced by `address(this)`.

Mellow: Acknowledged.

Cantina Managed: Acknowledged.

3.6.5 Centralization risk of `LpWrapper` contract

Severity: Informational

Context: LpWrapper.sol#L268

Description: The `LpWrapper` contract combines all the funds deposited by all users into a single liquidity position. The addresses possessing the `ADMIN_ROLE` and `ADMIN_DELEGATE_ROLE` of `LpWrapper` hold the right to change the position params of LpWrapper's position in `Core`.

The position params include:

- `StrategyParams`: parameters which determines the rebalancing strategy of entire pooled position:
  - which includes `StrategyType`, `tickNeighborhood`, `tickSpacing` & `width`.
- `CallbackParams`:
  - `gauge` - the Velo gauge to which the entire pooled position is deposited into.
  - `farm` - address to which all rewards are sent.
  - `counter` - contract which counts the sent rewards.
- `SecurityParams`: parameters which determine the price manipulation protection of position.

These parameters can be changed using the `LpWrapper.setPositionParams` function. The control over such sensitive parameters can pose a centralization risk for the users.

Recommendation: It is advised that a `Timelock` contract must be used to govern the admin-only features of `LpWrapper` contract.

Mellow: Acknowledged.

Cantina Managed: Acknowledged.

3.6.6 Inconsistent handling of `ammPositionIds` in `VeloDeployFactory.createStrategy`

Severity: Informational

Context: VeloDeployFactory.sol#L282

Description: In `VeloDeployFactory.createStrategy`, at VeloDeployFactory.sol#L282 the function tries to iterate over `ammPositionIds` array while a few statements back at VeloDeployFactory.sol#L231 the function itself declared the `ammPositionIds` as an array of length `1`. This behaviour is inconsistent and should be avoided.

Recommendation: Consider asserting that `ammPositionIds.length == 1` and read the `ammPositionIds[0]` value directly without iteration.

Mellow: Fixed in commit 736eef90.

Cantina Managed: The impacted code has been removed.

3.6.7 Invalid validation applied in `VeloOracle.getOraclePrice`

Severity: Informational

Context: VeloOracle.sol#L73-L74

Description: The `getOraclePrice` perform this check:

```
if (previousObservationIndex == observationCardinality) revert NotEnoughObservations();
```

As per the implementation of `CLPool` it can be observed that `observationIndex` will always be less than `observationCardinality`. Hence `previousObservationIndex` will always be less than `observationCardinality`. So the above mentioned validation check will always return `false`.

Recommendation: Consider removing the `revert` statement or convert it to an `assert` statement.

Mellow: Fixed commit 736eef90.

Cantina Managed: The impacted code segment has now been removed.

3.6.8 Shadowed function names

Severity: Informational

Context: LpWrapper.sol#L37-L43

Description: The `LpWrapper` constructor includes the parameters `name` and `symbol` which are used to provide to the inherited ERC20 contract constructor.

```
constructor(
    ICore core_,
    IAmmDepositWithdrawModule ammDepositWithdrawModule_,
    string memory name,
    string memory symbol,
    address admin
) ERC20(name, symbol) DefaultAccessControl(admin) {
```

These parameters, however, shadow the `name` and `symbol` functions in the ERC20 contract:

```
function name() public view virtual override returns (string memory) {
    return _name;
}

function symbol() public view virtual override returns (string memory) {
    return _symbol;
}
```

This doesn't pose any direct risk, but may cause problems later on with readability and maintainability.

Recommendation: Change the parameters to avoid shadowing the function names, e.g.:

```diff
  constructor(
      ICore core_,
      IAmmDepositWithdrawModule ammDepositWithdrawModule_,
-     string memory name,
+   string memory name_,
-     string memory symbol,
+   string memory symbol_,
      address admin
-   ) ERC20(name, symbol) DefaultAccessControl(admin) {
+   ) ERC20(name_, symbol_) DefaultAccessControl(admin) {
```

Mellow: Fixed in commit 831a70d3.

Cantina Managed: Issue is fixed as recommended.

3.6.9 Lack of input validation

Severity: Informational

Context: LpWrapper.sol#L64-L69, LpWrapper.sol#L194-L199

Description: `LpWrapper.deposit/withdraw` both do not validate that the amounts to deposit and withdraw are non-zero. We can see in the following proof of concepts, which can be added to `LpWrapper.t.sol`, that execution will successfully complete:

- `deposit`:

```solidity
function testDepositZero() external {
    pool.increaseObservationCardinalityNext(2);
    lpWrapper = new LpWrapper(
        core,
        depositWithdrawModule,
        "Wrapper LP Token",
        "WLP",
        Constants.OWNER
    );
    uint256 tokenId = mint(
        pool.token0(),
        pool.token1(),
        pool.tickSpacing(),
        pool.tickSpacing() * 20,
        10000,
        pool
    );
    uint256 positionId = _depositToken(tokenId, address(lpWrapper));
    lpWrapper.initialize(positionId, 10000);
    vm.startPrank(Constants.DEPOSITOR);
    deal(pool.token0(), Constants.DEPOSITOR, 1 ether);
    deal(pool.token1(), Constants.DEPOSITOR, 1 ether);
    IERC20(pool.token0()).approve(address(lpWrapper), 1 ether);
    IERC20(pool.token1()).approve(address(lpWrapper), 1 ether);
    vm.expectRevert(abi.encodeWithSignature("InsufficientLpAmount()"));
    lpWrapper.deposit(1 ether, 1 ether, 100 ether, Constants.DEPOSITOR);
    uint256 totalSupplyBefore = lpWrapper.totalSupply();
    IAmmModule.AmmPosition memory positionBefore = ammModule.getAmmPosition(
        tokenId
    );
    vm.stopPrank();
   lpWrapper.deposit(0, 0, 0, Constants.DEPOSITOR);
}
```

- `withdraw`:

```solidity
function testWithdrawZero() external {
    pool.increaseObservationCardinalityNext(2);
    lpWrapper = new LpWrapper(
        core,
        depositWithdrawModule,
        "Wrapper LP Token",
        "WLP",
        Constants.OWNER
    );
```

```
        uint256 tokenId = mint(
            pool.token0(),
            pool.token1(),
            pool.tickSpacing(),
            pool.tickSpacing() * 20,
            10000,
            pool
        );
        uint256 positionId = _depositToken(tokenId, address(lpWrapper));

        lpWrapper.initialize(positionId, 10000);

        vm.startPrank(Constants.DEPOSITOR);

        deal(pool.token0(), Constants.DEPOSITOR, 1 ether);
        deal(pool.token1(), Constants.DEPOSITOR, 1 ether);

        IERC20(pool.token0()).approve(address(lpWrapper), 1 ether);
        IERC20(pool.token1()).approve(address(lpWrapper), 1 ether);

        lpWrapper.deposit(1 ether, 1 ether, 0.1 ether, Constants.DEPOSITOR);

        uint256 totalSupplyBefore = lpWrapper.totalSupply();
        IAmmModule.AmmPosition memory positionBefore = ammModule.getAmmPosition(
            tokenId
        );

        uint256 depositorBalance = lpWrapper.balanceOf(Constants.DEPOSITOR);

        uint256 balance = lpWrapper.balanceOf(Constants.DEPOSITOR);

        vm.stopPrank();

        lpWrapper.withdraw(0, 0, 0, Constants.DEPOSITOR);
    }
```

Both of these functions have a very wide surface of execution which increases the risk involved with them being executed arbitrarily with 0 values.

Execution of both of these functions executes the `beforeRebalance` and `afterRebalance` hooks, similarly to execution of `Core.emptyRebalance`, which is a protected function. Further impact has not been discovered, but it's recommended that this is mitigated regardless.

Recommendation: Include logic that validates that the provided parameters are non-zero, e.g.:

```
if (lpAmount == 0 || minAmount0 == 0 || minAmount1 == 0) revert NON_ZERO_AMOUNTS();
```

Mellow: Acknowledged.

Cantina Managed: Acknowledged.

3.6.10 `VeloDeployFactory` `tickSpacing` collision possible in mappings

Severity: Informational

Context: VeloDeployFactory.sol#L13-L15

Description: `VeloDeployFactory` supports multiple pools which can have overlapping `tickSpacing` used for strategy creation. Prior to creating strategies with `createStrategy`, we set the strategy and deposit params according to the `tickSpacing` intended to be used with them during creation.

Updating strategy and deposit params according to `tickSpacing`:

```
/// @inheritdoc IVeloDeployFactory
function updateStrategyParams(
    int24 tickSpacing,
    StrategyParams memory params
) external {
    _requireAdmin();
    _tickSpacingToStrategyParams[tickSpacing] = params;
}

/// @inheritdoc IVeloDeployFactory
function updateDepositParams(
    int24 tickSpacing,
    ICore.DepositParams memory params
) external {
    _requireAdmin();
    _tickSpacingToDepositParams[tickSpacing] = params;
}
```

Retrieving strategy and deposit params in `createStrategy` according to the `tickSpacing` used:

```
StrategyParams memory strategyParams = _tickSpacingToStrategyParams[
    tickSpacing
];

// ...

ICore.DepositParams
    memory depositParams = _tickSpacingToDepositParams[tickSpacing];
```

Since multiple pools are supported which can have overlapping `tickSpacing`, we can run into a collision in setting strategy and deposit params for two different pools with the same `tickSpacing` where the second write will overwrite the first one, causing both strategies to be deployed with the overwritten deposit and strategy params.

Listing as informational severity since Mellow has indicated that usage will be aligned such that we don't run into this problem, which may only be caused by trusted actors.

Recommendation: Aligned with Mellow's indicated modified usage, internal facing documentation should be written that clearly indicates that this collision is possible and how to avoid it. Alternatively, a safer solution would be to use 3D mappings which include the pool address, e.g.:

```
mapping(address => mapping(int24 => IVeloDeployFactory.StrategyParams))
    private _tickSpacingToStrategyParams;
mapping(address => mapping(int24 => ICore.DepositParams)) private _tickSpacingToDepositParams;
```

Mellow: Fixed by modifying `VeloDeployFactory` to no longer need these variables.

Cantina: Issue is no longer present as a result of `VeloDeployFactory` changes. Fixed in commit 736eef90