yAudit Orange Finance Dopex V2 Automator Review

Review Resources:

- None beyond the code repositories

Auditors:

- engn33r
- spalen

Table of Contents

Review Summary

Orange Finance Dopex V2 Automator

Orange Finance Dopex V2 Automator provides an automated solution to manage deposits in Dopex CLAMM to maximize yield. The automation is performed by Gelato jobs, which allows the liquidity positions to be actively managed to stay within 2.5% of the current tick, making the Dopex positions eligible for Arbitrum STIP rewards. Users can deposit a single asset into the automator contract, which acts like an ERC4626 vault, although it is not compliant with the ERC4626 standard. When users redeem their shares in the vault, they do not receive only assets, because Dopex may not be able to return all funds in the case that the underlying assets are locked due to how the Dopex protocol operates. In this case, users can redeem their automator shares for assets and Dopex tokens.

The contracts of the Orange Finance Dopex V2 Automator https://github.com/orange-finance/dopex-v2-automator were reviewed over 14 days. The code review was performed by two auditors between December 15, 2023 and January 5, 2024. The repository was under active development during the review, but the review was limited to the latest commit at the start of the review. This was commit eee5932015036fa44593edf611a30bb99a52c883 for the Orange Finance repo.

Scope

The scope of the review consisted of the following contracts at the specific commit:

- OrangeDopexV2LPAutomator.sol
- OrangeDopexV2LPAutomatorV1Factory.sol
- StrategyHelper.sol

- lib/AutomatorUniswapV3PoolLib.sol
- lib/UniswapV3SingleTickLiquidityLib.sol

After the findings were presented to the Orange Finance team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Orange Finance and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Only a few functions are access controlled, and these functions have the proper access controls applied.
Mathematics	Average	Uniswap v3 tick math is used throughout this solution, which adds some complexity. Only basic math is used for accounting of contract balances.
Complexity	Average	The goals of the protocol are clear, but the logic necessary to build on top of a protocol that provides liquidity to Uniswap v3 (instead of adding liquidity directly to Uniswap v3) and managing multiple assets adds complexity to the design.
Libraries	Average	Some library dependencies are custom while others are slightly modified from Uniswap. The automator contract specifically has a large number of imported dependencies, adding to the complexity of the design. However, the imported libraries are well-known and considered safe.

Category	Mark	Description
Decentralization	Average	No contracts are deployed behind a proxy where code could be upgraded, but there are access controlled functions around rebalance and fee-setting functions. The Gelato jobs are managed off-chain and are naturally managed in a more centralized way by the owner of the Gelato jobs.
Code stability	Average	Although an earlier version of the code was deployed to mainnet with very small value for testing, the code changed since that deployment and the findings from this audit demonstrate some other changes are necessary before the code is ready for production.
Documentation	Low	No docs were provided about the protocol and some of the functions are missing comments and detailed NatSpec.
Monitoring	Low	There were no events to track the actions on-chain. The Gelato job dashboard does provide a good view of whether assets are rebalanced on schedule, but there is no clear visibility into whether the settings of the protocol are ideal (how much time is the current tick out of the active ticks range, is distributing assets equally across the ticks within 2.5% of the current tick the best approach vs. only allocating to ticks within 1% of the current tick, etc.).
Testing and verification	Average	The tests achieve 100% coverage of the core contracts OrangeDopexV2LPAutomator.sol, OrangeDopexV2LPAutomatorV1Factory.sol, and StrategyHelper.sol, but do not achieve as much coverage on the contracts in /lib and /vendor.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
    - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.

- Gas savings
  - Findings that can improve the gas efficiency of the contracts.
- Informational
  - Findings including recommendations and best practices.

Critical Findings

None.

High Findings

1. High - Uniswap spot price manipulation may cause loss of value

Under very specific circumstances, it may be possible to steal value from the automator contract because of how the Uniswap spot price is used in the calculation of `totalAssets()`. These circumstances may not occur on-chain, but should be considered.

Technical Details

There are two main ways that a user could steal value from a vault: one is to receive more shares during depositing than they should, and the other is to receive more assets during redeeming than they should. The first case, receiving more shares during depositing, is possible in OrangeDopexV2LPAutomator.sol. However, it may not be profitable except in certain cases. The manipulation is possible because in `deposit()`, shares are calculated with `convertToShares()`. `convertToShares()` divides by `totalAssets()`, so a lower `totalAssets()` value means the user gets more shares. The last line of `totalAssets()` uses `pool.currentTick()`, which uses slot0 (AKA spot price), which can be manipulated in a sandwich. So for this attack, the user wants to push the counter asset price down during the sandwiched `deposit()`. The sequence would look like:

1. To move the counter asset price down in the WETH-USDC pool, the user takes a WETH flashloan, then sells WETH to get USDC in the Uniswap pool
2. `deposit()` into automator using the lower value for the counter asset WETH, causing the vault to give more shares to the user than it should
3. User sells USDC to get WETH in the pool, and returns flashloan

4 User profit is the amount of extra shares they received minus the fees the user paid to Uniswap for the swaps that manipulated the pool spot price

No `rebalance()` needs to occur for this sandwich attack, because it is the reliance on `pool.currentTick()` in `OracleLibrary.getQuoteAtTick()` in the last line of `totalAssets()` that causes the issue.

Factors that would reduce the cost of this attack include:

- Greater liquidity in the automator contract, specifically in the form of counter assets, will increase the profit that the attacker can receive because the value is extracted from the `OracleLibrary.getQuoteAtTick()` calculation in `totalAssets()`.
- If the current tick drops below all the ticks where the automator contract has deposited assets into the pool, then the LP positions owned by the automator are 100% in the counterAsset (this is calculated by `LiquidityAmounts.getAmountsForLiquidity()`). For example, if you LP into the USDC-WETH pool at the current ticks and WETH price drops, you will hold 100% WETH. This scenario makes this attack even more profitable because the automator is holding mostly counterAssets.
- Even if there is a lot of liquidity in the Uniswap pool at a certain point in time, it is unclear who controls this liquidity. The user who wants to extract value from the automator vault may be able to withdraw (at least some of) this liquidity from Uniswap specifically to make price manipulation easier. Alternatively, if a user has deposited into the pool for the ticks that are passed through during the swap, then they are receiving the 0.05% Uniswap fee payment, so the swap gets closer to free the more liquidity the user has provided to Uniswap (to collect their own fees).
- The attack would have a lower cost if the liquidity at the current tick and nearby ticks are low, because this reduces the cost of changing the price beyond those ticks. At the time of this audit, the WBTC-USDC Uniswap v3 pool on Arbitrum only had $225k of liquidity while the Dopex WBTC-USDC market had $330k of liquidity. The scenario of lower liquidity around the current tick might be more likely during rapid price changes, because automated liquidity providers like the automator contract will not have time to increase the liquidity around the current tick.

Factors that would increase the cost of this attack include:

- When the automator contract has more assets and counter assets that would increase the profitability of this attack, the same assets and counter assets that are deposited into

the Uniswap v3 pool would make the cost of price manipulation higher

Impact

High. Loss of value may occur but only under very specific circumstances that are not always available on-chain.

Recommendation

Use Chainlink price data instead of the Uniswap pool spot price in `totalAssets()` and `freeAssets()` calculations. There is a Chainlink oracle for every asset currently supported by Dopex CLAMM:

- USDC/USD https://data.chain.link/arbitrum/mainnet/stablecoins/usdc-usd
- ETH/USD https://data.chain.link/arbitrum/mainnet/crypto-usd/eth-usd (not WETH, but no difference)
- WBTC/USD https://data.chain.link/arbitrum/mainnet/crypto-usd/wbtc-usd
- ARB/USD https://data.chain.link/arbitrum/mainnet/crypto-usd/arb-usd

If you want to avoid Chainlink dependency, you could use Uniswap TWAP but be careful of Uniswap pool liquidity, defined time period and the cost of price manipulation.

Developer Response

This issue has been confirmed and resolved with a recommended solution.

2. High - Wrong redeem calculation

In the redeem flow, the user will not get a portion of `counterAsset` for his shares.

Technical Details

In OrangeDopexV2LPAutomator shares represent four different assets in the contract:

- `asset` balance
- `counterAsset` balance
- redeemable shares in Dopex position
- locked shares in Dopex position

In the redeem flow, the user should get a portion of four assets. Redeemable shares and redeemed into `asset` or `counterAsset` depending on the current price. After that, all `counterAsset` is swapped for `asset` and it is sent to the user alongside the locked shares which cannot be redeemed at the moment.

The problem in calculating `counterAsset` user owns that will be swapped for asset. In this calculation, the portion of idle `counterAsset` is not accounted for, only `counterAsset` that is received from redeeming Dopex position.

Impact

High. The user will not get a portion of the currently idle counter asset on every redeem.

Recommendation

Account for the portion of user-owned idle `counterAsset` when redeeming.

```
- uint256 _payBase = counterAsset.balanceOf(address(this)) - _preBase;
+ uint256 _payBase = shares.mulDivDown(_preBase, _totalSupply) +
counterAsset.balanceOf(address(this)) - _preBase;
```

Developer Response

This issue has been confirmed and resolved with a recommended solution.

Medium Findings

1. Medium - Scheduled Gelato jobs could be frontrun

In the `rebalance()` function, OrangeDopexV2LPAutomator.sol uses swap slippage protection provided by StrategyHelper.sol in `swapParams`. Slippage protection is implemented by leaving 1% of all available assets free for swapping. Also, unused assets, both asset and counter asset, will be left idle instead of deposited into Dopex.

The gelato jobs are scheduled to run at set intervals making the transaction timing very predictable. The jobs are often executed within 1-2 seconds of the scheduled time interval. This could allow the Gelato job to be frontrun, and allow for a sandwich of the swap operation.

Technical Details

The Gelato job to call `checkLiquidizePooledAssets()` is scheduled to be called every 6 hours. `checkLiquidizePooledAssets()` uses a 1% slippage limit. If the Gelato job can be frontrun, which is not normally a concern in Arbitrum but could be in this case, then the 1% slippage value could be sandwiched, allowing another party to extract 1% of the value from the swap. If 1% of swap value disappears every 6 hours, this can negatively impact the users of the protocol.

Setting a constant 1% slippage limit will make `rebalance()` prone to sandwich attacks, especially for low liquidity pools like WBTC/USDC. The slippage fee is as constant 1% but some pools that are used have a 0.05% fee, so 1% slippage is far beyond what is needed and leaves value to be extracted by MEV. Calculating slippage protection is done off-chain and improving calculation won't add additional gas cost.

Impact

Medium. Frontrunning the job takes considerable effort and the profit that could be extracted will normally be quite low. However, if automated, 1% of the swapped value could be extracted every 6 hours when this job is run.

Recommendation

Use Uniswap current price or Chainlink Oracle to get the correct price of the asset. Use the precise amount of assets needed for a swap, this depends on the final implementation of `totalAssets()` function. If the fees needed for swapping counter asset to asset are not accounted for in `totalAssets()`, add this fee to the calculation. Finally, add a small slippage percentage for swapping to limit the possibility of failing transactions, and make it configurable in StrategyHelper. This enables fine-grained calculation for slippage protection amount.

Using the current price from Uniswap pool for calculating the maximum amount out for the swap would work in this flow because the `rebalance()` won't be called in the same transaction. `checkLiquidizePooledAssets()` is used off-chain by Gelato.

Developer Response

This issue has been mitigated by implementing slippage control variable.

2. Medium - Dopex can be paused and lock user funds

Dopex handler contract implements Pausable contract which could lock all user funds.

Technical Details

Dopex UniswapV3SingleTickLiquidityHandler can be paused which would disable all interactions with the contract. That means that all funds are locked, this includes the funds deposited by OrangeDopexV2LPAutomator.

If this happens, the user will not be able to withdraw the funds from OrangeDopexV2LPAutomator, even though there could be idle asset and counter funds that the user wants to redeem. This happens when the Dopex handler is paused and the automator has some redeemable shares. On every redeem, `_burnPosition()` call will revert, basically locking all funds.

Add test to file Redeem.t.sol to verify that burning redeemable shares will revert when Dopex is paused:

```
function test_redeem_revertDopexPaused() public {
    vm.startPrank(managerOwner);
    uniV3Handler.grantRole(uniV3Handler.PAUSER_ROLE(), managerOwner);
    vm.stopPrank();

    _depositFrom(bob, 1 ether);
    _depositFrom(alice, 1.3 ether);

    uint256 _balanceBasedWeth = WETH.balanceOf(address(automator));
    uint256 _balanceBasedUsdce = _getQuote(address(WETH), address(USDCE),
uint128(_balanceBasedWeth));

    (int24 _oor_belowLower, ) = _outOfRangeBelow(1);
    (int24 _oor_aboveLower, ) = _outOfRangeAbove(1);

    _mint(_balanceBasedWeth, _balanceBasedUsdce, _oor_belowLower, _oor_aboveLower);

    IUniswapV3SingleTickLiquidityHandler.TokenIdInfo memory _tokenIdInfo =
_tokenInfo(_oor_belowLower);
    uint256 _freeLiquidity = _tokenIdInfo.totalLiquidity - _tokenIdInfo.liquidityUsed;

    _useDopexPosition(_oor_belowLower, _oor_belowLower + pool.tickSpacing(),
uint128(_freeLiquidity - 1000));

    vm.prank(managerOwner);
    uniV3Handler.emergencyPause();

    vm.startPrank(alice);
    vm.expectRevert();
    (uint256 _assets, IOrangeDopexV2LPAutomator.LockedDopexShares[] memory _locked) =
automator.redeem(1.3e18, 0);
    vm.stopPrank();
}
```

Impact

Medium. When relying on other protocols, be aware of possible negative scenarios like pausing.

Recommendation

Before burning redeemable shares in Dopex, check if Dopex is paused. In the case it is paused, transfer redeemable shares to the user. This enables the user to redeem his shares without loss:

```
if (c.shareRedeemable > 0) {
    if (handler.paused()) {
        handler.safeTransferFrom(address(this), msg.sender, c.tokenId,
c.shareRedeemable, "");
    } else {
        _burnPosition(c.lowerTick, c.lowerTick + poolTickSpacing,
c.shareRedeemable.toUint128());
    }
}
```

Developer Response

This issue has been confirmed and resolved with a recommended solution.

3. Medium - `totalAssets()` returned value is too large

The value returned from `totalAssets()` and `freeAssets()` does not consider the liquidity in the Uniswap pool that can be used for swaps, nor does it consider the fees. The result is that `totalAssets()` will return a greater value than the real amount of assets returned if all assets owned by the contract were swapped and returned to users.

Technical Details

The last line of `totalAssets()` and `freeAssets()` calls `OracleLibrary.getQuoteAtTick()` using the pool's current tick. What this line of logic intends to do is to calculate how many assets the contract owns if all counterassets were swapped to assets. But if such a swap were to happen in reality, the value of `counterAssets` is not necessarily happening only at the current tick, because there is not infinite liquidity at this tick. Instead, when a swap function like `_swapToRedeemAssets()` or `_swapBeforeRebalanceMint()` is called, the call flow is `SwapRouter.exactOutputSingle()`/`SwapRouter.exactInputSingle()` -> `SwapRouter.exactOutputInternal()`/`SwapRouter.exactInputInternal()` -> `Pool.swap()`. In `Pool.swap()`, the swap loops through different ticks until the swap has been completed, which shows that if a tick has insufficient liquidity, the price of assets being swapped will not use the price only at one tick. This means that the value of counterAssets calculated by `OracleLibrary.getQuoteAtTick()` is only correct if the current tick has sufficient liquidity, and if the tick does not have sufficient liquidity, this value if an overestimation of how many assets can be received after swapping counterAssets.

`totalAssets()` also omits fees that are applied. The Uniswap v3 `Pool.swap()` operation does apply fees, which is 0.05% in the case of the USDC-WETH pool on Arbitrum.

The inaccurate `totalAssets()` return value is used directly by other logic in the protocol, specifically in `convertToShares()`, `convertToAssets()`, and `StrategyHelper.checkLiquidizePooledAssets()`. The side effects from `totalAssets()` returning an overestimated assets value is that `convertToShares()` will return an underestimated value, `convertToAssets()` will return an overestimated value, and `StrategyHelper.checkLiquidizePooledAssets()` will overestimate the `_distributableAssets` which increases the chance of a call to `checkLiquidizePooledAssets()` reverting due to smaller than expected slippage tolerance. The second order effects of this include `deposit()` returning lower than expected shares to depositors, because the number of shares is calculated by `convertToShares()` which underestimates the shares value.

The same pattern of using `OracleLibrary.getQuoteAtTick()` without considering fees is found in StrategyHelper.sol, in `getBurnableTicksInfo()` and `getAssetsPerLiquidityUnit()`. The result is a similar overestimation of assets.

Impact

Medium. The accounting of `totalAssets()` is inaccurate. If a user chooses to call `redeem()` with X shares, they will receive fewer assets than the value returned by `convertToAssets()` for the same number of shares (assuming all assets are redeemable and not locked in Dopex) because of the swap from counterAssets to assets. Gelato calls to `StrategyHelper.checkLiquidizePooledAssets()` are more likely to revert due to the overestimation of assets resulting in a small slippage tolerance than designed.

Recommendation

The existing logic to calculate `totalAssets()` is fine if the swap actions are removed. This primarily means replacing `_swapToRedeemAssets()` in `redeem()` with a transfer of `_payBase` counterAssets to msg.sender. Otherwise, the logic in `OracleLibrary.getQuoteAtTick()` is not designed to handle the realities of the on-chain state of the pool. The value returned by the Quoter contract's `quoteExactInputSingle()` or `quoteExactOutputSingle()` is more precise, but is not designed to be called on-chain.

Developer Response

This issue has not been addressed. This issue persists as long as automator takes single asset deposit/withdraw and when converting value of the both assets in one side since calling exactInputSingle() onchain is not feasible. The fact that user gets lower amount of assets when triggering redeem() compare to convertToAssets() is caused by the token swap from the counter asset to the asset to stick to single asset deposit/withdraw. Allowing redeem to return both assets without swapping will cause an attack vector and leaving from original concept of single asset deposit/withdraw, so we decided not to address this issue.

However, we have decided to do the following two things in response to the issue.

- Add a function that returns the Automator's positions. This allows StrategyHelper to add a function to calculate an accurate totalAsset() using exactInputSingle(). (assuming offchain calls)
- Change in vault design for fundamental solution of this issue in next version.

4. Medium - Uniswap dependencies don't use solidity 0.8.X release

Uniswap v3 is deployed on-chain with solidity 0.7.X, which does not include SafeMath. The math in the library was built around the assumption that SafeMath is not used, so a special 0.8.X compatible release was later created. Although this Dopex automator code is using

solidity 0.8.X, it is using the 0.7.X math assumptions from Uniswap, which will revert in overflow or underflow scenarios.

Technical Details

There is a special 0.8 branch of the Uniswap v3 core and periphery repositories. The uniswap v3 dependencies in package.json of dopex automator do not use the uniswap 0.8 branch code, meaning the old assumptions without SafeMath are applied. This is incorrect and will lead to code reverting in situations where it should not. For example, this math of `getLiquidityForAmount0()` is unchecked in the 0.8 branch of uniswap v3 periphery but is not unchecked in LiquidityAmounts.sol of dopex automator.

Impact

Medium. Math in solidity 0.8.X will revert in the case of overflow or underflow, which can prevent the protocol from functioning as expected in edge case scenarios.

Recommendation

Remove the Uniswap v3 npm dependencies in package.json. Instead, install the latest Uniswap v3 core and periphery solidity 0.8.X releases. This is the approach used by dopex CLAMM, as seen in the project's .gitmodules file. Replace the files in the contracts/vendor/uniswapv3 directory with library versions from the 0.8 release of uniswap v3.

Developer Response

This issue has been confirmed and resolved with a recommended solution.

5. Medium - Deposit limit should be applied to total asset value

Deposit cap is used to limit user deposit, but it is applied to a single deposit amount which can be bypassed by calling deposit multiple times.

Technical Details

Deposit cap is applied only to the current user deposit amount. This means that the cap is applied only to single deposit call which doesn't limit the user from depositing again and providing more assets above the limit.

Impact

Medium. Cap values should be enforced on total amounts to enable control of TVL in the contract.

Recommendation

Compare the deposit cap with the total asset value in the contract, not just the deposit input.

```
- if (assets > depositCap) revert DepositCapExceeded();
+ if (assets + totalAssets() > depositCap) revert DepositCapExceeded();
```

Developer Response

This issue has been confirmed and resolved with a recommended solution.

Low Findings

1. Low - Add additional swapping route

In `redeem()` and `rebalance()` function there is swapping between asset and counter asset. The protocol uses a naive approach that the Uniswap pool will provide the best quote for swapping all the time.

Technical Details

For swapping, only one Uniswap pool is used. This can lead to overpaying for swaps in `rebalance()` and `redeem()` calls because for larger amounts of swaps the end-user or the protocol will get less than if it would use other swapping routes.

For pair WBTC/USDC.e even swapping small amounts like 1000 USDC.e will get a better 0.25% rate on WooFiV2 compared to the used Uniswap pool. For higher amounts, the difference is even bigger 10000 USDC.e difference is 0.43%.

WETH/USDC.e pair will provide the best route using Uniswap pool for lower amounts until 50k USDC.e, after that WooFiV2 swap route is more efficient. Uniswap pair ARB/USDC.e will provide the best single route until 20k USDC.e amounts, for higher amounts WooFiV2 will give

better rates.

Impact

Low. Using only one swapping route could end up overpaying for swaps.

Recommendation

Implement additional routes for swapping assets. WooFiV2 will provide better quotes for higher amounts so it is a good alternative to implement alongside using Uniswap pool. Implementing another swap route for `rebalance()` is recommended because the quote is calculated off-chain and with minimal gas cost rebalancing will better swap rates. This will include changes to both `rebalance()` and `checkLiquidizePooledAssets()` but it will provide more efficiency to the protocol. As the protocol grows and adds more liquidity to Uniswap, it will be more efficient to use Uniswap pool, but on the other hand, a lot of that liquidity can be used in Dopex Option Markets which will reduce liquidity in Uniswap pool and better swap will be on other exchanges.

Using other exchanges for swapping will remove the problem of `rebalance()` swap changing the current tick and causing the revert when adding liquidity to the new current tick.

Developer Response

This issue has not been addressed. We will consider implementing offchain swap routing calculation for rebalance function from next version.

2. Low - Using tick to calculate price is less precise

Using sqrtPriceX96 is more precise than using the current tick when dealing with the current price of assets in a Uniswap v3 pool.

Technical Details

Uniswap docs indicate there is a difference between using sqrtPriceX96 and using the current tick for current asset price. sqrtPriceX96 is a more exact value than the current tick, but `OracleLibrary.getQuoteAtTick()` in `totalAssets()` uses the current tick. `OracleLibrary.getQuoteAtTick()` is also used in StrategyHelper.sol, in `getBurnableTicksInfo()` and `getAssetsPerLiquidityUnit()`.

Impact

Low. Slight inaccuracy will occur in asset conversion calculation.

Recommendation

Consider replacing `OracleLibrary.getQuoteAtTick()` with a more accurate method of calculation that uses sqrtPriceX96. A separate finding explains other inaccuracies that occur from `OracleLibrary.getQuoteAtTick()`.

Developer Response

This issue has been confirmed and resolved with a recommended solution.

3. Low - Choose `LIQUIDITY_UNIT` value for greater precision

`LIQUIDITY_UNIT` is a constant of value 1e9. The choice of this constant value is not explained and could be improved for greater rebalancing precision.

Technical Details

The value `LIQUIDITY_UNIT` has a comment of `magic number`, and the choice of this value is not further explained. The reason why the current value is a poor choice for the WETH-USDC pair is that the return value from `getAssetsPerLiquidityUnit()` when using variable length arrays filled with the current tick of the Uniswap v3 pool is on the order of magnitude of 1e2 or 1e3.

```
cast call 0x4849a93888222AC0743A00c532B93fBAAc245e0A "getAssetsPerLiquidityUnit(int24[])
(uint256)" "[-199220, -199220, -199220, -199220, -199220, -199220, -199220, -199220,
-199220, -199220, -199220, -199220, -199220, -199220, -199220, -199220, -199220, -199220,
-199220, -199220, -199220, -199220, -199220, -199220, -199220, -199220, -199220, -199220,
-199220, -199220]" --rpc-url https://arbitrum.llamarpc.com
```
Returns: 720

When dealing with assets and value, many smart contracts choose to have 1e18 decimals of precision. A difference of 1 in a value of 1e3 is roughly a 0.1% difference. The lack of precision is demonstrated when adding 1 in X “for safety reasons” (which is also not explained). The

result is that `_targetLiquidity` will suffer from lack of precision and could negatively impact the rebalancing process.

Impact

Low. Lack of precision in calculating target liquidity can lead to suboptimal asset allocation.

Recommendation

Use a value of `LIQUIDITY_UNIT` that is customized to the specific asset pair (WETH-USDC will differ from WBTC-USDC) such that `getAssetsPerLiquidityUnit()` returns a value that has greater precision, ideally a value of 1e10 or greater.

Developer Response

This issue has been confirmed and resolved by setting bigger number of unit (1e18) for presision.

4. Low - `rebalance()` function will revert when adding liquidity to the current tick

The Dopex handler doesn't support adding liquidity to the current tick. OrangeDopexV2LPAutomator `rebalance()` function expects the array of `ticksMint` without the current tick inside, but the current implementation doesn't guarantee that `ticksMint` doesn't contain the current pool tick.

Technical Details

StrategyHelper calculates `ticksMint` array for OrangeDopexV2LPAutomator and removes the current tick at the start of the calculation. It also calculates swap values, but with a naive approach that the swap won't change the current tick in the pool. The swap is done in `rebalance()` function before the liquidity is added to Dopex. If the swap changes the current tick, `ticksMint` will most likely contain the current tick and the call to mint position in Dopex will revert because Dopex doesn't support minting in the current pool tick.

Another more common problem is that the other transactions can do the swap and change the current tick in the pool. This is possible because the calculation and removal of the current is not done in the same transaction as `rebalance()` function. It was prepared before by Gelato and sent to `rebalance()` but the current pool tick could have changed before `rebalance()` was executed.

These are all possible cases because `ticksMint` array contains ticks +/- 2.5% from the current tick and any change from the calculated current tick will result in the new current tick ending in `ticksMint`.

Impact

Low. If the current pool tick is changed it will cause the `rebalance()` to revert when adding liquidity to Dopex to a new current tick. If `rebalance()` reverts, it will cost the protocol gas and the user will lose the opportunity to receive more STIP rewards from maintaining liquidity within the 2.5% range of the current tick.

Recommendation

Check the current pool tick inside `rebalance()` function and remove it from the mint list. Skip removing the current tick in StrategyHelper.

Developer Response

This issue has been confirmed and resolved with a recommended solution.

5. Low - Mint validity check doesn't cover all cases

OrangeDopexV2LPAutomator has the function to check mint validity which is used by StrategyHelper but it doesn't validate all mint ticks.

Technical Details

In StrategyHelper, check validity is done inside function `validateMintTicks()`. Mint ticks are validated only when determining all rebalance ticks. All rebalance ticks (mint, burn and stay) are combined into two new arrays, mint and burn info ticks. `classifyStayTicks()` function is combining these arrays.

The scenario that is not covered by the current implementation is when the `_stayTicksInfo` array contains the ticks that need more asset allocation. Allocating more assets is the same as minting, in Dopex it is `mintPositionHandler()`. This means that all ticks in `_newMintTicksInfo` must be validated, not just mint ticks.

Impact

Low. Missing mint validity could cause the `rebalance()` function to revert.

Recommendation

Validate ticks before adding them into the final mint array: `_newMintTicksInfo`.

While counting `mintExtensionCounter` validate ticks using automator function:

```
- } else if (_targetLiquidity > _stayTicksInfo[i].liquidity) {
+ } else if (_targetLiquidity > _stayTicksInfo[i].liquidity &&
automator.checkMintValidity(_stayTicksInfo[i].tick)) {
```

Also, while adding ticks to `newMintTicksInfo` validate ticks using automator function:

```
- } else if (_targetLiquidity > _stayTicksInfo[i].liquidity) {
+ } else if (_targetLiquidity > _stayTicksInfo[i].liquidity &&
automator.checkMintValidity(_stayTicksInfo[i].tick)) {
```

Developer Response

This issue has been confirmed and resolved with a recommended solution.

6. Low - Minting zero shares

The user can mint zero shares and lose the deposited assets if the OrangeDopexV2LPAutomator earns enough profit.

Technical Details

In the `deposit()` function there is a check for small deposits, but it doesn't guarantee that the user won't receive zero shares.

The number of shares is calculated by dividing the total shares by the total assets. If the OrangeDopexV2LPAutomator earns enough profit, or if tokens are airdropped into the contract, then the calculated shares for deposit will be zero for small amounts of assets and the user can get zero shares for their deposit.

Impact

Low. After the OrangeDopexV2LPAutomator earns enough profit, users can end up with zero shares.

Recommendation

Before minting new shares require that the amount of shares is more than zero. This is a standard check for ERC4626.

```
if (shares == 0) revert DepositTooSmall();
_mint(msg.sender, shares);
```

Developer Response

This issue has been confirmed and resolved with a recommended solution.

7. Low - Avoid reentrancy in `deposit()` and `redeem()`

Common implementations of ERC4626 use a specific sequence of minting shares and transferring assets in `deposit()` to prevent ERC777 reentrancy. Even if no ERC777 is planned to be supported by the protocol, there is no harm in following best practices for the `deposit()` and `redeem()` functions.

Dopex positions are ERC1155 which are transferred as locked positions in `redeem()` function. ERC1155 tokens use `onERC1155Received()` which opens up the risk of reentrancy attacks.

Technical Details

In `deposit()` of OrangeDopexV2LPAutomator.sol, assets are transferred as the final action in the function, after shares are minted. This is the opposite of the ERC4626 `deposit()` function implemented in OpenZeppelin and solmate, which transfer the assets before minting shares.Those libraries even include a comment on which this sequence is chosen, which is to prevent reentrancy from a ERC777 or similar token.

The `redeem()` implementation uses a burn-then-transfer sequence for the asset and shares, which does match the implementation in OpenZeppelin and solmate. However, this burn-then-transfer sequencer is not followed in `redeem()` for the locked dopex shares in this for loop, which are transferred then burned. Dopex tokens are ERC1155 tokens, which call the recipient's `onERC1155Received()` function after ERC1155 token balances are updated. This opens an opportunity for reentrancy because the Automator contract shares are not burned until the very end of the `redeem()` function, which this call allows ERC1155 reentrancy.

Impact

Low. Although Dopex is not expected to support any ERC777 token, it is best to be safe and follow the best practices for `deposit()` and `redeem()` to avoid any reentrancy risks. Dopex locked position tokens are ERC1155 and have a hook `onERC1155Received()` that should be avoided.

Recommendation

Follow the "Checks Effects Interactions" pattern and update the internal state in `deposit()` and `redeem()` before transferring tokens. Move token burning to the top of the `redeem()` function. Also, move the `transferFrom()` function in `deposit()` to the beginning of the function, after the deposit cap check. Note that additional changes are needed in calculating the share amount.

Alternatively, use a reentrancy guard to protect the functions from being called again (see OpenZeppelin ReentrancyGuard.sol).

Developer Response

This issue has been confirmed and resolved with a former recommended solution.

8. Low - No check when setting `minDepositAssets`

The value of `minDepositAssets` is set by the admin, but there are no checks around whether the chosen value is within an acceptable range.

Technical Details

There are some assumptions around the value of `minDepositAssets` that would be best checked directly in the smart contract logic.

1. If `minDepositAssets` is less than or equal to the number of dead shares that are minted on the first deposit, then the first depositor could receive zero shares for their deposit. There is no check in `deposit()` to prevent minting zero shares to the depositor (unlike `redeem()`, which explicitly checks for a shares value of zero), most likely because it is assumed that `minDepositAssets` is set properly. Even if `minDepositAssets` is equal to the number of minted dead shares, the current check of reverting if `assets < minDepositAssets` would still allow `assets = minDepositAssets` which would still mint zero shares.
2. There is no check to prevent a `_fee` of zero in `deposit()`. If there is a concern that a user could deposit many small amounts to bypass the fee, then in order to prevent `shares.mulDivDown(depositFeePips, 1e6)` from rounding down to zero, `shares * depositFeePips >= 1e6` must be true. If shares are assumed to be minted in a 1-to-1 ratio

with assets, and if it is assumed that depositFeePips can be as low as 1 without allowing the user to pay no fees, then this simplifies to `assets >= 1e6`. So `minDepositAssets` should not be less than 1e6, otherwise it may be possible (depending on the value of `depositFeePips`) for a user to pay zero fees on their deposit.

Impact

Low. `deposit()` could return zero shares or the user could pay zero fees depending on the value of `minDepositAssets`.

Recommendation

Consider adding checks as needed in the constructor to ensure the immutable `minDepositAssets` is set properly. Note that these suggestions may only make sense with USDC as the chosen `asset`, but may not make as much sense if a high value token such as WBTC is the chosen `asset`.

```
require(minDepositAssets_ > (10 ** IERC20Decimals(address(asset_)).decimals() / 1000));
if (minDepositAssets_ < 1e6) revert;
```

Developer Response

This issue has been confirmed and resolved with a recommended solution.

9. Low - First depositor loses some value from dead shares

The first depositor loses a small amount of value due to the logic in `deposit()`.

Technical Details

The first depositor will lose 10 ** (decimals - 3) of value compared to later depositors. For an asset of USDC, this equals only $0.001 worth of value, but the penalty is unevenly applied only to the first depositor. The cause is the line `shares = assets - _dead`, which reduces the shares received by the first depositor, in contrast to later deposits where shares are minted at a 1-to-1 ratio to deposited assets.

Consider a first depositor who deposits 1e7 of USDC tokens ($10) and then wants to immediately withdraw. Ignoring any fees, the user will deposit 1e7 of tokens and receive 1e7 - 1e3 = 9999000 shares out of a total of 1e7 shares (because 1e3 shares are dead). When the user immediately redeems their shares, they will only receive 1e7 - 1e3 USDC, so 1e3 USDC got locked in the contract. This may be a small amount of value for USDC, but if WBTC was

the asset in another automator contract, the dead shares would be 1e5 worth of WBTC, or $42 at the current $42,000 price of WBTC. This amount of value loss could be more significant for the first depositor.

Impact

Low. The chosen solution to prevent inflation attacks results in a small penalty for the first depositor.

Recommendation

To prevent greater loss of value in other asset pairs, always choose the lower value asset in the asset pair (normally USDC) to be the `asset` token in the contract. For example, in a ETH/USDC pair, USDC should be the underlying asset because 1 ETH has more value than 1 USDC.

Additionally, the protocol team should consider depositing the first liquidity into a newly created contract so that the small value loss is accepted by the protocol, rather than by normal users.

Developer Response

Dev team will deposit first, so this issue isn't addressed.

10. Low - Incorrect decimals used for max fee constant

The comment for `MAX_PERF_FEE_PIPS` indicates the max fee should be 1%, but the actual value of the `MAX_PERF_FEE_PIPS` constant is 10%.

Technical Details

The comment above `MAX_PERF_FEE_PIPS` states `max deposit fee percentage is 1%` but the value of this constant is 100_000, or 1e5. If 1e5 correlates to 1%, then 1e7 should correlate to 100%. But elsewhere in the code, the deposit fee is divided by 1e6 (not 1e7) and a different comment also suggests that 1e6 correlates to 100%.

Impact

Low. Incorrect decimals can allow the admin to set a higher fee than intended.

Recommendation

Change the value of `MAX_PERF_FEE_PIPS`.

```
- uint24 constant MAX_PERF_FEE_PIPS = 100_000;
+ uint24 constant MAX_PERF_FEE_PIPS = 10_000;
```

Developer Response

This issue has been confirmed and resolved with a recommended solution.

11. Low - Limit the size of active ticks set

A set of active is used to iterate in a for loop to track assets. If the size of the set grows too big, it will block functionalities by consuming all gas inside a `for` loop.

Technical Details

Strategist can change the set size by calling `rebalance()` function. If the size of the active ticks set grow too big, it could unable the user from calling `redeem` function and locking their funds.

Impact

Low. Only guarded calls can grow the size of the set.

Recommendation

Calculate the maximum set size that will enable the user to call the `redeem()` function without experiencing out of gas problems.

Developer Response

This issue has been confirmed and resolved with a recommended solution.

12. Low - Utilization of stay ticks not considered in `classifyStayTicks()`

`classifyStayTicks()` determines whether stay ticks should be added to the mint or burn array. If a stay tick is added to the burn array, the Dopex liquidity will be burned in order to recover the underlying assets. But the utilization of the liquidity is not considered, meaning that it is possible that `rebalance()` will try to burn liquidity that is utilized, and because the liquidity cannot be burned while it is utilized, `rebalance()` will revert.

Technical Details

Dopex options can be minted for liquidity provided at any tick, regardless of whether the tick is near or far from the current tick. A liquidity provider can only withdraw their liquidity from Dopex when no option is written against it. This means that during `rebalance()`, any tick in the array of ticks to burn should have zero utilization. A check is performed in `getBurnableTicksInfo()`, where `automator.getTickFreeLiquidity()` is called to retrieve the available liquidity that can be burned from the Dopex handler contract. But there is no comparable check performed on stay ticks that are later classified as burn ticks inside of `classifyStayTicks()`, meaning that stay ticks that are added to the list of burn ticks may not be unutilized. If `rebalance()` attempts to burn liquidity that is utilized, the call will revert because that liquidity cannot be burned while it is utilized.

Impact

Low. If the stay tick is added to the burn array but has utilization in Dopex, it will cause the `rebalance()` to revert because not all the liquidity can be burned. If `rebalance()` reverts, it will cost the protocol gas and the user will lose the opportunity to receive more STIP rewards from maintaining liquidity within the 2.5% range of the current tick.

Recommendation

A simple approach to avoid a revert in `rebalance()` would be to add a check in `classifyStayTicks()` to avoid adding a stay tick to the burn array if the utilization is above 0.

More complicated solutions are possible to provide more optimal capital efficiency. In short, the assets that can be reallocated should be moved to active ticks that are in the 2.5% range of the current tick. Assets that can be reallocated includes ticks with liquidity outside of the active ticks and ticks that are inside the active ticks but with too many assets, and in these two cases, only the liquidity in Dopex that is unutilized can be burned. This is easy to do for liquidity outside of the active ticks range, where all unutilized liquidity can be burned to retrieve the underlying assets. To calculate how to allocate the liquidity of the stay ticks, consider an approach such as:

1. calculate what the average liquidity would be by combining the burned liquidity from outside the active ticks and the assets in the active ticks.
2. for any active ticks that have too many assets, leave it as is (it is still earning the same 2.5% rewards as long as it is in the active range), but subtract the amount of assets over the target amount from the liquidAssets (a new variable that acts like totalAssets, but it only counts assets that can be reallocated).
3. Recalculate the average liquidity using the new liquidAssets.
4. Mint more liquidity in the active ticks appropriately so that every active tick reaches the average liquidity amount.

Developer Response

This issue has been confirmed and resolved with another way after discussion with auditor.

Gas Saving Findings

1. Gas - Duplicate zero case logic for `convertToShares()`

`convertToShares()` has logic to handle the case of `totalSupply == 0`, but the one place that calls `convertToShares()` also has logic for this case.

Technical Details

`convertToShares()` contains logic to handle the case of `totalSupply == 0`, but there is also logic to handle this case inside `deposit()`, which is the only place in the automator contract that calls this function. The logic inside `convertToShares()` is not strictly needed for the proper functioning of the smart contract, and because the automator is not trying to comply with the EIP4626 spec, no problem exists with compliance to a spec.

Impact

Gas savings.

Recommendation

Consider removing the duplicate logic in `convertToShares()` for a small gas savings.

```
    function convertToShares(uint256 assets) public view returns (uint256) {
-       uint256 _supply = totalSupply;
-
-       return _supply == 0 ? assets : assets.mulDivDown(_supply, totalAssets());
+       return assets.mulDivDown(totalSupply, totalAssets());
    }
```

Developer Response

This issue has been confirmed and resolved with a recommended solution.

2. Gas - Store immutable pool fee value

Uniswap pool fee can be stored as an immutable variable to save gas cost on every swap.

Technical Details

Uniswap pool `fee` is immutable variable but is fetched in every swap. Storing this value in the construct will reduce the gas cost of future swaps.

Impact

Gas savings.

Recommendation

Add the immutable variable `poolFee` and set it once in the constructor. Use it for all swaps instead of calling `pool.fee()` L504, L591 and L606.

Gas savings data from provided tests.

`test_freeAssets_noDopexPosition() (gas: -88 (-0.025%))`

`test_redeem_dopexPositionPartiallyLocked() (gas: -580 (-0.033%))`

`test_redeem_burnDopexPositions() (gas: -1095 (-0.065%))`

`test_rebalance_fromInitialState() (gas: -1381 (-0.097%))`

`test_integration_checkLiquidizePoolAsset() (gas: -65577 (-0.257%))`

`test_integration_opsProxyExec() (gas: -65687 (-0.271%))`

`test_createOrangeDopexV2LPAutomator_revertNotAdmin() (gas: -32100 (-0.470%))`

`test_createOrangeDopexV2LPAutomator_roleGranted() (gas: -32100 (-0.478%))`

`test_createOrangeDopexV2LPAutomator_onlyInit() (gas: -96991 (-0.807%))`

`test_scenario_some_none_below_t1up_none_mint() (gas: -64979 (-0.851%))`

`test_scenario_some_some_above_t0up_deposit_mint() (gas: -66295 (-0.898%))`

`test_deposit_deductedPerfFee_secondDeposit() (gas: -64890 (-1.135%))`

`test_deposit_deductedPerfFee_firstDeposit() (gas: -64956 (-1.168%))`

`test_deposit_firstTime() (gas: -64957 (-1.183%))`

`test_calculateRebalanceSwapParamsInRebalance_reversedPair() (gas: -64979 (-1.199%))`

`test_totalAssets_reversedPair() (gas: -64979 (-1.201%))`

`test_getActiveTicks() (gas: -64846 (-1.202%))`

`test_freeAssets_reversedPair() (gas: -65090 (-1.203%))`

`test_deposit_revertWhenDepositCapExceeded() (gas: -64979 (-1.204%))`

`test_deposit_revertWhenDepositTooSmall() (gas: -64979 (-1.232%))`

`Overall gas change: -1010859 (-0.616%)`

Developer Response

This issue has not been addressed. We will implement this change in next version.

3. Gas - Remove AccessControlEnumerable if only one role is used

OrangeDopexV2LPAutomatorV1Factory implements AccessControlEnumerable but uses only one role DEFAULT_ADMIN_ROLE.

Technical Details

OrangeDopexV2LPAutomatorV1Factory implements OZ contract AccessControlEnumerable which provides fine grained access control for multiple roles. The OrangeDopexV2LPAutomator.sol contract uses only one role, so there is no need to import a multiple-role OZ contract. It is cheaper to implement a simple immutable admin address for restricting access to the function `createOrangeDopexV2LPAutomator()` or import the simpler OZ Ownable contract.

Impact

Gas savings.

Recommendation

Consider removing the AccessControlEnumerable import from OrangeDopexV2LPAutomatorV1Factory. Replace it with an immutable variable `ADMIN` which is set in the constructor and used to restrict the access to function `createOrangeDopexV2LPAutomator()`. If there is a need to change the admin address later and an immutable address will not work, consider using the OZ Ownable import instead of AccessControlEnumerable.

Developer Response

This issue has not been addressed. We will implement this change in next version.

4. Gas - Use unchecked for gas savings

Unchecked math can be used when there is no risk of overflow or underflow due to existing logic preventing such conditions.

Technical Details

Unchecked math can be used in a few places, including:

- subtraction in `deposit()`
- subtraction for counterAssets in `redeem()`
- subtraction and addition for assets in `redeem()`

Impact

Gas savings.

Recommendation

Use unchecked when no risk of underflow exists.

Developer Response

This issue has not been addressed. We will implement this change in next version.

5. Gas - Cache `pool_.token0()` in OrangeDopexV2LPAutomator constructor

Technical Details

`pool_.token0()` is an external call that happens at least twice in the constructor (1, 2) for OrangeDopexV2LPAutomator.sol. Caching this value will save roughly 1438 gas on every Automator deployment. `pool_.token1()` can be cached too, but provides a much smaller gas improvement, because it is not guaranteed to be called twice in every constructor call.

Impact

Gas savings.

Recommendation

Make the following changes to the constructor of OrangeDopexV2LPAutomator.sol.

```
+ address token0 = pool_.token0();
+ address token1 = pool_.token1();
- if (asset_ != IERC20(pool_.token0()) && asset_ != IERC20(pool_.token1())) revert
TokenAddressMismatch();
+ if (asset_ != IERC20(token0) && asset_ != IERC20(token1)) revert
TokenAddressMismatch();

manager = manager_;
handler = handler_;
router = router_;
pool = pool_;
asset = asset_;
- counterAsset = pool_.token0() == address(asset_) ? IERC20(pool_.token1()) :
IERC20(pool_.token0());
+ counterAsset = token0 == address(asset_) ? IERC20(token1) : IERC20(token0);
```

The gas savings from `forge snapshot --diff` with only `token0` cached:

```
test_deposit_firstTime() (gas: -1438 (-0.026%))
test_calculateRebalanceSwapParamsInRebalance_reversedPair() (gas: -1438 (-0.027%))
test_freeAssets_reversedPair() (gas: -1438 (-0.027%))
test_totalAssets_reversedPair() (gas: -1438 (-0.027%))
test_deposit_revertWhenDepositCapExceeded() (gas: -1438 (-0.027%))
test_getActiveTicks() (gas: -1439 (-0.027%))
test_deposit_revertWhenDepositTooSmall() (gas: -1438 (-0.027%))
test_createOrangeDopexV2LPAutomator_onlyInit() (gas: -42169 (-0.351%))
test_createOrangeDopexV2LPAutomator_revertNotAdmin() (gas: -40730 (-0.596%))
test_createOrangeDopexV2LPAutomator_roleGranted() (gas: -40730 (-0.607%))
Overall gas change: -133696 (-0.081%)
```

The gas savings from `forge snapshot --diff` with `token0` and `token1` cached:

```
test_deposit_firstTime() (gas: -1454 (-0.026%))
test_calculateRebalanceSwapParamsInRebalance_reversedPair() (gas: -1454 (-0.027%))
test_freeAssets_reversedPair() (gas: -1454 (-0.027%))
test_totalAssets_reversedPair() (gas: -1454 (-0.027%))
test_deposit_revertWhenDepositCapExceeded() (gas: -1454 (-0.027%))
test_getActiveTicks() (gas: -1454 (-0.027%))
test_deposit_revertWhenDepositTooSmall() (gas: -1454 (-0.028%))
test_createOrangeDopexV2LPAutomator_onlyInit() (gas: -60844 (-0.506%))
test_createOrangeDopexV2LPAutomator_revertNotAdmin() (gas: -59390 (-0.869%))
test_createOrangeDopexV2LPAutomator_roleGranted() (gas: -59390 (-0.885%))
Overall gas change: -189802 (-0.116%)
```

Developer Response

This issue has not been addressed. We will implement this change in next version.

6. Gas - Remove unused code

Some code is never used and can be removed to save gas during deployment.

Technical Details

Functions that are declared but never used include:

- `mulDivRoundingUp()` in FullMath.sol

- `consult()` in OracleLibrary.sol
- `getChainedPrice()` in OracleLibrary.sol
- `getTickAtSqrtRatio()` in TickMath.sol

Impact

Gas savings.

Recommendation

Remove or comment out the unused functions.

Developer Response

The contract import is changed from local to library, so cannot address this issue.

7. Gas - Use minimal number of external calls

In function `totalAsset()` Uniswap pool `slot0` if fetched two times. Using only one call will save gas.

Technical Details

Uniswap pool `slot0` is fetched first time at L192 and second time at L229 `pool.currentTick()`. It is a view function, the value of the pool tick cannot change so it will be more gas-efficient to remove the second call to pool and get the tick value in the first call.

Impact

Gas savings.

Recommendation

Change function `totalAssets()` as following:

```
function totalAssets() public view returns (uint256) {
    // 1. calculate the total assets in Dopex pools
    uint256 _length = activeTicks.length();
    uint256 _tid;
    uint128 _liquidity;
    (int24 _lt, int24 _ut) = (0, 0);
    (uint256 _sum0, uint256 _sum1) = (0, 0);

    (uint160 _sqrtRatioX96, int24 _tick, , , , , ) = pool.slot0();

    for (uint256 i = 0; i < _length; ) {
        _lt = int24(uint24(activeTicks.at(i)));
        _ut = _lt + poolTickSpacing;
        _tid = handler.tokenId(address(pool), _lt, _ut);

        _liquidity = handler.convertToAssets((handler.balanceOf(address(this),
_tid)).toUint128(), _tid);

        {
            (uint256 _a0, uint256 _a1) = LiquidityAmounts.getAmountsForLiquidity(
                _sqrtRatioX96,
                _lt.getSqrtRatioAtTick(),
                _ut.getSqrtRatioAtTick(),
                _liquidity
            );
```

```
            _sum0 += _a0;
            _sum1 += _a1;
        }
        unchecked {
            i++;
        }
    }

    // 2. merge into the total assets in the automator
    (uint256 _base, uint256 _quote) = (counterAsset.balanceOf(address(this)),
asset.balanceOf(address(this)));

    if (address(asset) == pool.token0()) {
        _base += _sum1;
        _quote += _sum0;
    } else {
        _base += _sum0;
        _quote += _sum1;
    }

    return
        _quote + // quote == asset balance
        OracleLibrary.getQuoteAtTick(_tick, _base.toUint128(), address(counterAsset),
address(asset));
}
```

Gas savings data from provided tests.

`test_totalAssets_hasDopexPositions() (gas: -1384 (-0.113%))`

`test_redeem_dopexPositionPartiallyLocked() (gas: -2786 (-0.158%))`

`test_redeem_burnDopexPositions() (gas: -2786 (-0.166%))`

`test_rebalance_fromInitialState() (gas: -3458 (-0.242%))`

`test_redeem_dopexPositionFullyLocked() (gas: -2792 (-0.262%))`

`test_deposit_secondTime() (gas: -1399 (-0.277%))`

`test_totalAssets_noDopexPosition() (gas: -1399 (-0.400%))`

`test_redeem_noDopexPosition() (gas: -2798 (-0.512%))`

`test_redeem_revertWhenSharesTooSmall() (gas: -2797 (-0.518%))`

`test_redeem_revertWhenMinAssetsNotReached() (gas: -2799 (-0.535%))`

`test_convertToShares_hasDopexPositions() (gas: -13924 (-0.857%))`

`test_convertToAssets_hasDopexPositions() (gas: -15322 (-0.937%))`

`test_convertToShares_noDopexPosition() (gas: -8391 (-1.185%))`

`test_convertToAssets_noDopexPosition() (gas: -9789 (-1.365%))`

Developer Response

This issue has not been addressed. We will implement this change in next version.

8. Gas - Always add new ticks to EnumerableSet

It is cheaper to call EnumerableSet add function every time instead of checking if the value exits because the same check is implemented inside the library.

Technical Details

OZ library for `EnumerableSet._add()` will verify that value is not stored before storing it. This means the additional check in the code is not needed before adding to EnumerableSet. Removing this check will save gas.

Impact

Gas savings.

Recommendation

Remove unneeded check at L551.

```
- _posId = uint256(keccak256(abi.encode(handler, pool, _lt, _ut)));
- if (handler.balanceOf(address(this), _posId) == 0)
activeTicks.add(uint256(uint24(_lt)));
+ activeTicks.add(uint256(uint24(_lt)));
```

Gas savings data from provided tests. Savings will be higher with more active ticks to iterate inside for loop.

```
test_redeem_dopexPositionFullyLocked() (gas: -1160 (-0.109%))
test_redeem_dopexPositionPartiallyLocked() (gas: -2396 (-0.136%))
test_redeem_burnDopexPositions() (gas: -2320 (-0.138%))
test_convertToAssets_hasDopexPositions() (gas: -2472 (-0.151%))
test_convertToShares_hasDopexPositions() (gas: -2473 (-0.152%))
test_getTickFreeLiquidity() (gas: -2280 (-0.172%))
test_rebalance_activeTickRemoved() (gas: -1121 (-0.174%))
test_freeAssets_hasDopexPositions() (gas: -2242 (-0.182%))
test_totalAssets_hasDopexPositions() (gas: -2243 (-0.183%))
test_getTickAllLiquidity() (gas: -2396 (-0.201%))
test_checkMintValidity_false_owed1Invalid() (gas: -48 (-0.217%))
test_checkMintValidity_true() (gas: -48 (-0.218%))
test_checkMintValidity_false_owed0Invalid() (gas: -48 (-0.218%))
test_validateMintTicks() (gas: -480 (-0.277%))
test_rebalance_fromInitialState() (gas: -4051 (-0.283%))
test_checkBurnOuterLiquidity() (gas: -4800 (-0.285%))
test_createOrangeDopexV2LPAutomator_revertNotAdmin() (gas: -54570 (-0.799%))
test_createOrangeDopexV2LPAutomator_roleGranted() (gas: -54570 (-0.813%))
test_createOrangeDopexV2LPAutomator_onlyInit() (gas: -104963 (-0.874%))
test_deposit_firstTime() (gas: -50392 (-0.918%))
test_calculateRebalanceSwapParamsInRebalance_reversedPair() (gas: -50392 (-0.930%))
test_freeAssets_reversedPair() (gas: -50392 (-0.931%))
test_totalAssets_reversedPair() (gas: -50392 (-0.931%))
test_deposit_revertWhenDepositCapExceeded() (gas: -50392 (-0.934%))
test_getActiveTicks() (gas: -50396 (-0.934%))
test_deposit_revertWhenDepositTooSmall() (gas: -50392 (-0.956%))
Overall gas change: -597429 (-0.364%)
```

Developer Response

This issue has not been addressed. We will implement this change in next version.

9. Gas - Fetch multiple liquidities from Dopex in one call

UniswapV3SingleTickLiquidityLib can be extended to gas-optimize fetching liquidity data from the handler.

Technical Details

OrangeDopexV2LPAutomator fetches `redeemableLiquidity` and `lockedLiquidity` using UniswapV3SingleTickLiquidityLib. Data for both liquidities can be calculated by fetching data from handler one once.

Impact

Gas savings.

Recommendation

Extend UniswapV3SingleTickLiquidityLib with one additional function:

```
    function liquidities(
        IUniswapV3SingleTickLiquidityHandler handler,
        address owner,
        uint256 tokenId_
    ) internal view returns (uint256, uint256) {
        uint256 _shares = handler.balanceOf(owner, tokenId_);
        IUniswapV3SingleTickLiquidityHandler.TokenIdInfo memory _tki =
handler.tokenIds(tokenId_);
        uint256 _maxRedeem = handler.convertToAssets(uint128(_shares), tokenId_);
        uint256 _freeLiquidity = _tki.totalLiquidity - _tki.liquidityUsed;

        uint256 _myMaxFreeLiquidity = Math.min(
            _maxRedeem,
            _freeLiquidity
        );

        if (_freeLiquidity >= _maxRedeem) return (_myMaxFreeLiquidity, 0);

        return (_myMaxFreeLiquidity, _maxRedeem - _freeLiquidity);
    }
```

Use this new function in OrangeDopexV2LPAutomator:

```
-            c.shareRedeemable = uint256(
-               handler.convertToShares(handler.redeemableLiquidity(address(this),
c.tokenId).toUint128(), c.tokenId)
-            ).mulDivDown(shares, _totalSupply);
-            c.shareLocked = uint256(
-                handler.convertToShares(handler.lockedLiquidity(address(this),
c.tokenId).toUint128(), c.tokenId)
-            ).mulDivDown(shares, _totalSupply);
+           (uint256 redeemableLiquidity, uint256 lockedLiquidity) =
handler.liquidities(address(this), c.tokenId);
+           c.shareRedeemable = uint256(
+               handler.convertToShares(redeemableLiquidity.toUint128(), c.tokenId)
+           ).mulDivDown(shares, _totalSupply);
+           c.shareLocked = uint256(
+                handler.convertToShares(lockedLiquidity.toUint128(), c.tokenId)
+           ).mulDivDown(shares, _totalSupply);
```

Gas savings data from provided tests. Savings will be higher with more active ticks to iterate inside for loop.

`test_redeem_dopexPositionFullyLocked() (gas: -7582 (-0.712%))`

`test_redeem_dopexPositionPartiallyLocked() (gas: -15180 (-0.859%))`

`test_redeem_burnDopexPositions() (gas: -15185 (-0.902%))`

Developer Response

This issue has not been addressed. We will implement this change in next version.

Informational Findings

1. Informational - Optimize calling `rebalance()`

OrangeDopexV2LPAutomator `rebalance()` is an expensive function to call. It is triggered by Gelato checking `checkLiquidizePooledAssets()` which always returns true. Checking the profitability and returning `false` when it's not profitable could run down the cost of maintaining the protocol.

Technical Details

The function `rebalance()` is by far the most expensive call in the protocol. From the test data, the average gas cost is 1.84M, a maximum of 12.2M. The protocol should strive to optimise this function, but with the current design, it is called every six hours without any checks. By checking one `rebalance()` call on-chain, it is clear that it consumes even more gas, almost 15.5M. The protocol should keep in mind that Gelato takes an additional 10% of that gas cost.

When there is a big deposit, or many small ones in a short period, there is no check or trigger to call `rebalance()`. Waiting for six hours to deposit could be costly for the end users because they are missing out on STIP rewards which is the main benefit of this strategy.

Impact

Informational.

Recommendation

It is recommended to make additional checks in `checkLiquidizePooledAssets()` before returning `true` and triggering `rebalance()` call.

A simple check if `_distributableAssets` is below the configurable variable `minDepositAssets` would make the protocol more profitable with minimal effort:

```
  uint256 _distributableAssets = automator.totalAssets() -
_totalUnburnableValueInAssets;
+ if (_distributableAssets < minDepositAssets) {
+     return (false, bytes("not enough assets"));
+ }
```

Another check can be added for `newMintTicksInfo`, to enforce that the size of the array must be above zero.

These are just the simple checks, the protocol should explore how `rebalance()` behaves on-chain and explore additional gas cost reductions without sacrificing optimal asset allocation. With these checks, the Gelato task can be reconfigured to call `rebalance()` after `checkLiquidizePooledAssets()` returns true. It could be called more often and be profitable if the `minDepositAssets` is set correctly. Or it will wait longer than a fixed six-hour schedule if there is no asset to deploy.

Yearn V3 uses multiple checks and triggers for strategies, check it out for more improvements.

Developer Response

This issue has been addressed as recommended solution after StrategyHelper revamping.

2. Informational - Skip calculating swap data when it is not needed

The function `checkBurnOuterLiquidity()` calculates rebalance swap data but it's not needed because that only burns liquidity won't swap any tokens.

Technical Details

Swap data is calculated using automator function `calculateRebalanceSwapParamsInRebalance()` has input params `ticksMint` and `ticksBurn`. For burning outer liquidity, `tickMint` is an empty array which will result with zero values for `_mintAssets` and `_mintCAssets`. This means that asset shortage will always be zero. In automator `rebalance()` that uses rebalance swap params, just skips swapping when asset shortage is zero.

Impact

Informational.

Recommendation

Simplify code by removing calculating rebalance swap data when burning outer liquidity:

```
- IOrangeDopexV2LPAutomator.RebalanceSwapParams memory _swapParams = automator
-     .calculateRebalanceSwapParamsInRebalance(_mintTicksInfo, _burnTicksInfo);
+ IOrangeDopexV2LPAutomator.RebalanceSwapParams memory _swapParams;
```

Developer Response

We call this function only offchain. This issue has not been addressed.

3. Informational - Withdrawing tokens

OrangeDopexV2LPAutomator contract is actively handling the Dopex position, which could be rewarded with additional tokens. Not having an option to withdraw will lock these tokens in the contract.

Technical Details

Addresses that actively manage funds in the DeFi can be rewarded with additional funds. OrangeDopexV2LPAutomator contract could also receive some tokens in the future. Adding the withdrawal function could enable the protocol and the users to earn additional funds.

Impact

Informational.

Recommendation

Add withdraw function, but don't enable withdrawing `asset` and `counterAsset` tokens.

```
function withdraw(IERC20 token) external onlyRole(DEFAULT_ADMIN_ROLE) {
    if (token == asset) revert TokenNotPermitted();
    if (token == counterAsset) revert TokenNotPermitted();
    token.safeTransfer(msg.sender, token.balanceOf(address(this)));
}
```

Developer Response

This issue has been resolved with a recommended solution.

4. Informational - NatSpec typos

Some NatSpec comments have typos.

Technical Details

1. A comment in StrategyHelper.sol mentions types of rebalance strategies, but there are only two.
2. There is an incomplete sentence in the NatSpec for `getCurrentLT()`.
3. `analyzeTickChanges()` has a comment suggesting the validity of ticks are checked, but this is done with a separate call to `validateMintTicks()` so that portion of the comment should be deleted.

Impact

Informational.

Recommendation

Fix the typos. Modify the first comment as follows:

```
-     * @notice Three type of rebalance() strategy.
+     * @notice There are two types of rebalance() strategies.
```

Developer Response

Confirmed, and resolved.

5. Informational - `tickRange` should be immutable

The `tickRange` variable is a function of the pool tick spacing. `poolTickSpacing` is an immutable variable that is set in the constructor, so `tickRange` should be set in the same way.

Technical Details

The three existing Dopex CLAMM markets have a tick spacing of 10, but if StrategyHelper.sol wishes to support a dopex CLAMM market that does not have a tick spacing of 10, the hard-coded constant value of 25 for `tickRange` will be incorrect. There is a comment for `tickRange` stating that the value is only correct `when tickSpacing is 10`, so this value should not be hard-coded.

The tick spacing of the existing markets can be checked on-chain:

- Dopex WETH-USDC market -> 10 tickSpacing
- Dopex WBTC-USDC market -> 10 tickSpacing
- Dopex ARB-USDC market -> 10 tickSpacing

Impact

Informational.

Recommendation

Consider changing `tickRange` to an immutable variable that is set in the constructor, even though the development team has clarified that the dopex contracts that will be supported are the 3 markets mentioned above that all have 10 tickSpacing.

Developer Response

We implemented a setter function for tickRange for future update of strategy.

6. Informational - Unnecessary import can be removed

The ERC20 import in IOrangeDopexV2LPAutomator.sol and StrategyHelper.sol is unnecessary.

Technical Details

The solmate ERC20 import in IOrangeDopexV2LPAutomator.sol and StrategyHelper.sol is unnecessary and can be removed.

Impact

Informational.

Recommendation

Remove the solmate ERC20 import in IOrangeDopexV2LPAutomator.sol and StrategyHelper.sol.

Developer Response

This issue has been resolved.

7. Informational - No dead shares for tokens with 2 decimals

The protective measure used to prevent an inflation attack on the first deposit only works when the decimals of the asset token is 3 or more. An asset with decimals of 2 will not mint any dead shares, meaning the inflation protection logic will not do anything.

Technical Details

The number of dead shares to mint is calculated by this line with `uint256 _dead = 10 ** decimals / 1000;`. If the result of `10 ** decimals` is less than 1000, then the division will round down and `_dead` will be zero. This means dead shares are only minted when the decimal value of the asset is 3 or greater, and an asset with a decimals value of 2 such as GUSD will not mint any dead shares.

Impact

Informational.

Recommendation

It is unlikely for Dopex CLAMM to add support for a token with 2 decimals, but it might be worth adding a check in the constructor similar to `require(address(asset_)).decimals() > 2)` to verify that this protective measure will always be enabled. If the assumption is made that no tokens with less than 3 decimals will be supported as an asset, then this line can be unchecked for gas savings.

Developer Response

This issue has been resolved.

8. Informational - Minor NatSpec improvements

Some NatSpec comments can be improved.

Technical Details

The NatSpec can be improved in some minor ways.

- `rebalanceSwapParams` has no named return variable in `calculateRebalanceSwapParamsInRebalance()` and the struct type is spelled with a capital 'R' as `RebalanceSwapParams`
- `checkMintValidity()` doesn't have any NatSpec for the `lowerTick` argument
- There is no NatSpec describing the function arguments for `analyzeTickChanges()` in StrategyHelper.sol

Impact

Informational.

Recommendation

Consider improving NatSpec as suggested.

Developer Response

This issue has been resolved.

9. Informational - `_setupRole()` and `safeApprove()` are deprecated

Deprecated functions from OpenZeppelin library imports are used, but it is a best practice to replace these with functions that will remain in future versions of the library.

Technical Details

To promote future compatibility, `_grantRole()` is recommended instead of `_setupRole()`. `_setupRole()` has been deprecated and starting from 5.X, `_setupRole()` no longer exists in the AccessControl library.

A similar suggestion applies to `safeApprove()`, which was deprecated in 2020. `safeApprove()` still exists in OZ 4.X versions but is removed in OZ 5.X.

Impact

Informational.

Recommendation

Avoid using deprecated functions when possible. Replace `_setupRole()` with `_grantRole()`. Replace `safeApprove()` with `safeIncreaseAllowance()`.

Developer Response

This issue has been resolved.

10. Informational - Missing event emits

Some functions that transfer values or modify state variables do not have events. Events can assist with analyzing the on-chain history of contracts and are therefore beneficial to add in important functions.

Technical Details

OrangeDopexV2LPAutomator updates the storage variable but doesn't emit events. Functions that could have events added include:

- `setDepositCap()`
- `setDepositFeePips()`
- `deposit()`
- `redeem()`
- `rebalance()`

Impact

Informational.

Recommendation

Add events to the functions listed above.

Developer Response

This issue has been resolved.

Final remarks

The design of the Orange Finance Dopex v2 automator differs from automated liquidity management of LP positions directly because the addition of Dopex CLAMM in between the automator and the Uniswap pool. This adds complexity to the design. The design also borrows from ERC4626 but has many structural differences to many standard ERC4626 tokens, such as returning multiple assets to users upon redemption.

One of the difficulties introduced by this complexity is determining the value of assets held by the vault, especially during `deposit()` and `redeem()` calls. Any imprecise accounting during `deposit()` or `redeem()` can lead to incorrect value being allocated to users.

A sizeable amount of the logic in this solution happens off-chain, which could not be fully reviewed. This includes the Gelato jobs to automatically allocate assets to active ticks that receive Arbitrum STIP rewards as well as the logic involved in claiming the Arbitrum STIP rewards through https://merkl.angle.money/. For example, the Merkl docs indicate that positions under $20 may not be eligible to receive rewards, but it is unclear whether this is applied (and how it is calculated) for the specific case of Arbitrum STIP rewards in Dopex v2.

The team is recommended to explore the possibility of using Yearn V3 tokenized strategy which can abstract away the complexity of ERC4626 and rebalance logic. This will also allow the team to leverage the Yearn ecosystem and potentially attract more users to the automator.