yAudit Dopex V2 CLAMM Review

Review Resources:

- Design document
- Dopex Docs

Auditors:

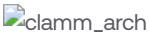
- usmannk
- spalen
- popular

Table of Contents

Review Summary

Dopex V2 CLAMM

Dopex V2 CLAMM provides a solution to solve the on-chain options liquidity issue, by using Uniswap V3 liquidity positions as collateral for writing options. With this approach, an option is purchased at a specific strike price against a Uniswap V3 LP previously deposited into CLAMM. The liquidity is borrowed from the corresponding tick and withdrawn from the AMM. The tokens are no longer participating in liquidity provision, and the CLAMM LPs are paid a fixed premium base which is converted into an LP position. When the current price surpasses the strike price, Traders can take a profit, in this case, the unwrapped liquidity is swapped into an asset, and the profit is sent to the trader, and the rest is LPed back to the AMM. The premium paid for this operation is far more than the fees the position might have earned while in the AMM. If the Uniswap V3 liquidity position are not used by any options, the liquidity position will still earn fees as it normally would in Uniswap V3.



The contracts of the Dopex V2 CLAMM Repo were reviewed over 21 days. The code review was performed by 3 auditors between October 20 and November 10, 2023. 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 ceb91c7d403da4a3b3eea831c195305e6a5362f9 for the Dopex V2 CLAMM repo.

Scope

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

- src/handlers/UniswapV3SingleTickLiquidityHandler.sol
- src/swapper/SwapRouterSwapper.sol
- src/uniswap-v3/LiquidityManager.sol
- src/DopexV2OptionPools.sol
- src/DopexV2PositionManager.sol

After the findings were presented to the Dopex V2 CLAMM team, fixes were made and included in several commits.

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, Dopex V2 CLAMM and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Average	Some functions have access control modifiers for certain roles to perform special functions. Modifiers appear to be used properly and are not missing or overused.
Mathematics	Average	There is no complex math, only internal accounting math. Uniswap Q notation is used for price handling.

Category	Mark	Description
Complexity	Average	The protocol relies on user depositors and traders to match to enable higher earnings for both sides. No complexity but the user should be guided to provide liquidity in specific liquidity ranges.
Libraries	Good	Only OpenZeppelin and Uniswap V3 external libraries were imported. Additional code was borrowed from Uniswap V3 for adding liquidity.
Decentralization	Low	The protocol does not have decentralized governance yet and the protocol team still controls many key protocol values and can withdraw all the funds from the protocol.
Code stability	Average	Due to tight timelines, the code did not receive an ideal level of test coverage. The protocol has a well-defined flow, but some optimizations could be made in calls between different contracts.
Documentation	Good	The development team provided a document about system design and possible problems. Proper NatSpec and detailed comments existed on the functions that needed explaining.
Monitoring	Average	Events were emitted where applicable but missing in some functions. Indexing values would be useful.
Testing and verification	Low	Only a few tests were available at the time of the review.

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

1. Critical - Failure to consume full allowance when minting leads to DoS

When minting positions via the `DopexV2PositionManager`, the token allowances required for the specified liquidity amount are granted via OpenZeppelin's `safeApprove()`. This function is deprecated. If the existing allowance is non-zero, `safeApprove()` will revert. A user can craft a call to `DopexV2PositionManager#mintPosition()` such that not all the allowance is consumed, and all future calls to `mintPosition()` for a given token will revert.

Technical Details

The issue lies in the usage of `safeApprove()` in `DopexV2PositionManager#mintPosition()`. It is currently possible for any user to mint an in-range position such that not all the tokens approved are consumed. All subsequent calls to `token.safeApprove()` would thus revert.

The following functions in `DopexV2PositionManager` use `safeApprove()` and would be rendered unusable:

- `mintPosition()`
- `unusePosition()`
- `donateToPosition()`

```
    function mintPosition(IHandler _handler, bytes calldata _mintPositionData)
        onlyWhitelistedHandlers(_handler)
    {
```

```
        // snip

        (address[] memory tokens, uint256[] memory amounts) = _handler.tokensToPullForMint(_mintPositionData);

        for (uint256 i; i < tokens.length; i++) {
            IERC20(tokens[i]).safeTransferFrom(msg.sender, address(this), amounts[i]);
            // @audit next line will revert if the handler's existing approval is nonzero
            IERC20(tokens[i]).safeApprove(address(_handler), amounts[i]);
        }
        sharesMinted = _handler.mintPositionHandler(msg.sender, _mintPositionData);
    }
```

Proof of concept:

```
// Drop into DopexPositionManager.t.sol
function testMintPositionAtk() public {
    uint256 amount0 = 5e18;
    uint256 amount1 = 5e18;

    // Range that contains the current tick
    int24 tickLower = -76040;
    int24 tickUpper = -75990;
    (uint160 sqrtPriceX96, int24 tick,,,,,) = pool.slot0();

    console2.log("tickLower sqrtRatio: ", TickMath.getSqrtRatioAtTick(tickLower));
    console2.log("tickUpper sqrtRatio: ", TickMath.getSqrtRatioAtTick(tickUpper));
    console2.log("current sqrtRatio:", sqrtPriceX96);
    console2.log("current tick: %s", tick); // -76010

    console.log("token0 allowance before: ", token0.allowance(address(positionManager), address(uniV3Handler)));
    console.log("token1 Allowance before: ", token1.allowance(address(positionManager), address(uniV3Handler)));

    // Mint the position
    positionManagerHandler.mintPosition(token0, token1, amount0, amount1, tickLower, tickUpper, pool, bob);

    console.log("token0 allowance after: ", token0.allowance(address(positionManager), address(uniV3Handler)));
    console.log("token1 Allowance after: ", token1.allowance(address(positionManager), address(uniV3Handler)));

    vm.expectRevert("SafeERC20: approve from non-zero to non-zero allowance");
    positionManagerHandler.mintPosition(token0, token1, amount0, amount1, tickLower, tickUpper, pool, bob);
    vm.expectRevert("SafeERC20: approve from non-zero to non-zero allowance");
    positionManagerHandler.mintPosition(token0, token1, amount0, amount1, tickLower, tickUpper, pool, jason);
}
```

Note that the proof of concept above is non-exhaustive. There are additional ways to cause lingering approvals, e.g. by returning two different prices from the user-controlled pool address in the two calls to `_getCurrentSqrtPriceX96()` in the `mintPosition()` flow.

Impact

Critical. Any user is able to render the system largely unusable.

Recommendation

Avoid using the deprecated `safeApprove()`, opting instead for `safeIncreaseAllowance()` and `safeDecreaseAllowance()`.

Developer Response

Fix - https://github.com/dopex-io/dopex-v2-clamm/pull/1/commits/8495b4127f9435b237c2563dcf17aff22b0efd94

High Findings

1. High - Incorrect conversion from call asset to quote asset

Converting the price per call asset in quote asset units is incorrect for `sqrtPriceX96` above `type(uint128).max`.

Technical Details

The function `_getPrice()` returns the price of call asset in quote asset. It returns the correct values as long `sqrtPriceX96` is below `type(uint128).max`.

When the price is above, calculating `priceX192` is incorrect because the amount is `priceX128`. The calculation multiples `sqrtPriceX96` with itself to get `priceX192`, but it also divides the value with `1 << 64` which lowers the precision to `priceX128 = priceX192 / X64`. After confirming that the calculated price is in X128 and not in X192 precision, converting the price to quote asset must be changed. To get the correct price precision, the value `1 << 128` must be used instead of `1 << 192` here and here.

Impact

High. All pools with `sqrtPriceX96` above `type(uint128).max` will have incorrect prices in quotes.

Recommendation

Use price precision for `sqrtPriceX96` above `type(uint128).max`.

```
function _getPrice(
    IUniswapV3Pool _pool,
    uint160 sqrtPriceX96
) internal view returns (uint256 price) {
    if (sqrtPriceX96 <= type(uint128).max) {
        uint256 priceX192 = uint256(sqrtPriceX96) * sqrtPriceX96;
        price = callAsset == _pool.token0()
            ? FullMath.mulDiv(
                priceX192,
                10 ** ERC20(callAsset).decimals(),
                1 << 192
            )
            : FullMath.mulDiv(
                1 << 192,
                10 ** ERC20(callAsset).decimals(),
                priceX192
            );
    } else {
        uint256 priceX128 = FullMath.mulDiv(
            sqrtPriceX96,
            sqrtPriceX96,
            1 << 64
        );

        price = callAsset == _pool.token0()
            ? FullMath.mulDiv(
                priceX128,
                10 ** ERC20(callAsset).decimals(),
                1 << 128
            )
            : FullMath.mulDiv(
                1 << 128,
                10 ** ERC20(callAsset).decimals(),
                priceX128
            );
```

```
        }
    }
```

Developer Response

Fix - https://github.com/dopex-io/dopex-v2-clamm/pull/1/commits/c128738324d4c139b7b25a9a12d6a5ff909055f8 and https://github.com/dopex-io/dopex-v2-clamm/pull/1/commits/0aaaff5865d045cf2ad09639ed9e5ecd27a62dfa

Medium Findings

1. Medium - Maximum slippage allowed

In UniswapV3SingleTickLiquidityHandler compounding liquidity pool fees is done by swapping token0 fees to token1 with maximum slippage allowed.

Technical Details

Swapping fees is done without any slippage protection. The team specified in the design document that they are aware of MEV attacks but don't see it as a problem but define it as "auto-compounding cost". It's defined as a game theory problem where MEVs will race each other to the bottom to get minimal profit. Additionally, minting new positions will auto-compound fees and lower slippage problems by calling the swap more frequently.

This theory assumes that minting new positions will be frequent enough to lower the profitability of MEV attacks. It is a big assumption to take, and it is advised to lower this attack surface.

Impact

Medium. MEV can take fees earned by users.

Recommendation

There are a few possibilities to lower the MEV attack:

1. Use Chainlink Oracle to have a non-manipulated asset price.
2. Define the maximum amount that can be used in a single swap. This is not an ideal solution, but it can eliminate the need for frequent new mints by users. `UniswapV3SingleTickLiquidityHandler` handles multiple pools and assets, so it should have a mapping of assets to maximum amounts. These values should be changeable by the owner.

Developer Response

As per outlined in the design doc, we don't consider this as a very big issue, as it allows MEV bots to race against to extract as much profit as they can, while being the first to extract it. We choose this design over others for multiple UX and Integration reasons. We don't depend on the frequent user deposits or withdrawals, rather we rely on the robust MEV ecosystem, that would want to extract value whenever possible and as fast as possible.

This is a similar assumption taken by most of the protocols in the space. We define it as "auto-compounding cost".

2. Medium - User deposits can be locked for a long time

DopexV2OptionPools uses user deposits to create options by pulling deposits. These deposits will be locked in:

1. If the settler won't call the settle function, user funds are locked
2. When an expired option is inside the liquidity range.

Technical Details

After the option is created, DopexV2OptionPools holds user deposits. The deposits can be returned by calling `exerciseOptionRoll()` by position owner or `settleOptionRoll()` which is limited only to settlers. If the settlers addresses won't call the function, user deposit will stay locked inside DopexV2OptionPools contract.

The second, less common case is when user deposits can get locked, settlers are working, but the option is inside the liquidity pool range. This comes as an error by the user when defining too wide a liquidity range. Now attackers can lock user deposits by creating an option with the highest possible range and minimal TTL to pay the minimum premium. Because the option cannot be settled while it's inside the range, user deposit is locked until the position gets out of the range.

Impact

Medium. A user must wait for settlers to call the settle function. Also, a user must be careful when defining the liquidity range.

Recommendation

Change function `settleOptionRoll()` to be permissionless, so anyone can settle options and return deposits. If the protocol wants to keep guarding this call only for settlers it can be done by adding additional time after which anyone can call the function. This way settlers have priority, but there is no fear that the deposits will stay locked.

The guard for the second case is to limit the maximum range that can be used when creating the option. This will limit the attackers from creating unsettable options and keep the user deposit safe and earning swap fees for their liquidity. Maximum range value can be defined as variable, so it can be changed by the contract owner if needed.

Developer Response

1) We have a periphery that can allow users to settle after a set period of time - https://github.com/dopex-io/dopex-v2-clamm/blob/feat/events/src/periphery/OpenSettlement.sol

2) Users should not deposit more than one tick liquidity. Our UI only allows one-tick deposits. As discussed, in the future update, I'm planning to provide a way for the user to donate and unlock liquidity if they have LPed in a long range. However, we don't ever recommend LPing at a long range.

3. Medium - Insecure calculation of premium donate liquidity

To mint an option, the user must pay the premium. This value is converted to pool liquidity, but the user can manipulate calculation of pool liquidity amount.

Technical Details

When minting the call option, the user defines params `OptionRollParams` for both upper and lower tick. Additionally, the user defines ticks for each pool he will use. Tick values in `OptionRollParams` are used to verify that all pools will have the same upper tick, for call option.

The user has the option to define a lower tick which is used only for calculating the liquidity amount needed to donate. By changing this value, user can manipulate to calculation to get lower liquidity amount which donated to liquidity pools. Even though the user has lowered the liquidity amount, he will pay the full premium amount and the excess amount will stay in the contract. Users that provided liquidity are taking the loss here.

Test `testBuyCallOptionRoll()` is providing an example of a minting call option. If the `tickLower` is changed from `tickLowerCalls` to `tickLowerCalls - 5000`, `amountLToDonate` that goes to liquidity providers is lowered from `452684065811520200 9043` to only `10211905461192628493`, which is 443 times less liquidity for users.

Impact

Medium. The user manipulating liquidity calculation cannot extract the value, but it will cause loss value for users that provide liquidity.

Recommendation

Define only one tick in `OptionRollParams` which will be upper or lower depending on bool `isCall`. Use `getAmount0ForLiquidity` or `getAmount1ForLiquidity` instead of `getLiquidityForAmounts`. Also, define only one tick in the stored struct `OptionData`.

Developer Response

Fix - https://github.com/dopex-io/dopex-v2-clamm/commit/c92e7b4ee64eb792988473985fcc83a0cab0b36b

Low Findings

1. Low - Using `block.number` for tracking passed time

UniswapV3SingleTickLiquidityHandler is using `block.number` to track the time between donations. Different L2 chains use different values and timespan for this value.

Technical Details

Tracking time between donations is done by storing `block.number`. Different L2 chains use different approach for value `block.number`:

- Arbitrum uses the value of L1, synced every minute. This means that the Arbitrum block number is updated on every 4th L1 block. Example from Aribtrum docs:

Wall Clock time	12:00 AM	12:00:15 AM	12:00:30 AM	12:00:45 AM	12:01 AM	12:01:15 AM
L1 block.number	1000	1001	1002	1003	1004	1005
L2 block.number	1000	1000	1000	1000	1004	1004
Arbitrum Block number (from RPCs)	370000	370005	370006	370008	370012	370015

It can cause a loss for a user if he burns the position on 3rd block because shares are converted to asset and donation liquidity is tracked using block number. Default lock duration is set to 100 which means that the user can lose up to 4% of donation.

- Optimism uses the value of its own L2 block number which is updated every 2 seconds, but it may change in the future. This implementation won't cause a loss of donation liquidity for the user.
- zkSync was using L1 block number, but it has switched to returning L2 block number for `block.number`. Chains can switch the implementation and this must be verified before deploying to the specific chain.

Impact

Low. The user could lose a part of the donation liquidity.

Recommendation

Because the first deployment is planned for Arbitrum, it's recommended to change from using `block.number` to `block.timestamp` to remove possible losses for the users on Arbitrum.

The choice of `block.timestamp` or `block.number` should be carefully considered for every chain that the code is deployed to.

Developer Response

Yes, we decided to go with block.number for all the chains for uniformity, and accept the few block interest lost if the user wants to withdraw early. In a way, it incentivizes users to stay till X block.

2. Low - Missing pool validation

DopexV2OptionPools are created with immutable tokens. The owner can whitelist new pools and change the prime pool without verifying that it uses the same tokens.

Technical Details

In the constructor, prime pool is set without verifying that is the correct one. Also, pools can be approved without verification if they use correct tokens.

Impact

Low. Only the contract owner can approve new pools.

Recommendation

Validate that approved pools have the correct token pair. In the constructor:

```
        primePool = IUniswapV3Pool(_primePool);
        if (primePool.token0() != _callAsset && primePool.token1() != _callAsset)
            revert DopexV2OptionPools__InvalidPool();
        if (primePool.token0() != _putAsset && primePool.token1() != _putAsset)
            revert DopexV2OptionPools__InvalidPool();
```

And in the function `updateAddress()`:

```
        IUniswapV3Pool pool = IUniswapV3Pool(_pool);
        if (pool.token0() != callAsset && pool.token1() != callAsset)
            revert DopexV2OptionPools__InvalidPool();
        if (pool.token0() != putAsset && pool.token1() != putAsset)
            revert DopexV2OptionPools__InvalidPool();
```

Developer Response

Fix - https://github.com/dopex-io/dopex-v2-clamm/pull/1/commits/6bd18992396562867 45266367ec9afabaca23e81

Gas Saving Findings

1. Gas - Skip sending zero amounts

Position manager is used to transfer the tokens between handler and option pool but in some cases, it will call transfer for amount zero. Skipping unneeded calls will save gas.

Technical Details

- `mintPosition()` calls transfer for all provided tokens, but the handler expects to receive only one token and fails if two tokens are sent.
- `usePosition()` and `unusePosition()` calls transfer for all tokens, but only one token is approved to position manager for transfer in option pools contract. That means only one token will be transferred from the position manager.
- `donateToPosition()` call transfer from on all tokens from the list, but it gets approved only token from option pools contract.

Additionally, some tokens will revert when sending zero amounts which would block the protocol from using these tokens.

Impact

Gas savings.

Recommendation

With the current implementation of DopexV2OptionPools, only one token is transferred from position manager. Skip sending amounts zero can save gas for the current implementation.

Suggestion for donate to position which can be applied to other specified functions:

```
- for (uint256 i; i < tokens.length; i++) {
-     IERC20(tokens[i]).safeTransferFrom(msg.sender, address(this), a[i]);
-     IERC20(tokens[i]).safeApprove(address(_handler), a[i]);
- }
+ uint256 amount;
+ for (uint256 i; i < tokens.length; i++) {
+     amount = a[i];
+     if (amount != 0) {
+         IERC20(tokens[i]).safeTransferFrom(msg.sender, address(this), amount);
+         IERC20(tokens[i]).safeApprove(address(_handler), amount);
+     }
+ }
```

Gas savings data from provided tests:

`testBurnPosition() (gas: -22233 (-1.507%))`
`testSettleOptionPutITMRoll() (gas: -28542 (-1.922%))`
`testExercisePutOptionRoll() (gas: -28542 (-1.934%))`
`testUsePosition() (gas: -32642 (-1.946%))`
`testSettleOptionCallITMRoll() (gas: -28542 (-1.949%))`
`testExerciseCallOptionRoll() (gas: -28542 (-1.961%))`
`testMintPositionWithSwaps() (gas: -27792 (-1.990%))`
`testSplitPosition() (gas: -27296 (-2.308%))`
`testUnusePosition() (gas: -54544 (-2.588%))`
`testBuyPutOptionRoll() (gas: -24796 (-2.609%))`
`testPutOptionSim() (gas: -40488 (-2.854%))`
`testCallOptionSim() (gas: -40488 (-2.855%))`
`testBuyCallOptionRoll() (gas: -27296 (-2.939%))`
`testDonation() (gas: -52264 (-3.458%))`

`testSettleOptionPutOTMRoll() (gas: -36642 (-3.555%))`

`testSettleOptionCallOTMRoll() (gas: -39142 (-3.883%))`

`testMintPosition() (gas: -27792 (-4.116%))`

`Overall gas change: -567583 (-2.287%)`

Developer Response

Fix - https://github.com/dopex-io/dopex-v2-clamm/pull/4/commits/5a9c540d26241c9d4a5320549b020defa7193060 and https://github.com/dopex-io/dopex-v2-clamm/pull/4/commits/523d500729090d7fbce12d8618781d80324614 08

2. Gas - Cache values outside the loop

Some values can be stored in variables before loops to reduce the gas cost compared to fetching the same value inside each iteration.

Technical Details

In function `exerciseOptionRoll()` token address to use is calculated in each iteration on multiple spots here, here and so on. These values will be the same for all iterations because the value is fetched outside the loop and will be the same for all option ticks. Extracting two addresses will reduce gas costs and improve readability.

The same can be applied in function `settleOptionRoll()`.

In both functions, variable `isAmount0` is defined inside a loop but can be calculated only once outside the loop because it is dependent on the `primePool` which won't change.

Impact

Gas savings.

Recommendation

Extract two addresses to variables `assetToUse` and `assetToGet`:

```
ERC20 assetToUse = ERC20(oData.isCall ? callAsset : putAsset);
ERC20 assetToGet = ERC20(oData.isCall ? putAsset : callAsset);
```

Move `isAmount0` outside the loops.

Developer Response

Fix - https://github.com/dopex-io/dopex-v2-clamm/pull/1/commits/46e38a7e72a97e82e4f0f3fe794cedd9c7daacca and https://github.com/dopex-io/dopex-v2-clamm/pull/1/commits/a728f8a25d35179df74d7740d1b3d71659225dde

3. Gas - Store immutable token decimals

In DopexV2OptionPools contract, token decimals are fetched multiple times in a few functions. These values won't change in the contract, and they can be stored to save gas.

Technical Details

Token addresses: `callAsset` and `putAsset` cannot be changed in the contract and can be set as immutable. Decimals of those tokens are fetched multiple times, e.g. here and here. Because token addresses cannot change, token decimals cannot also change and should be stored to save gas on each call.

Impact

Gas savings.

Recommendation

Add immutable variables to store `callAsset` and `putAsset` decimals and set the values in the constructor.

Developer Response

Fix - https://github.com/dopex-io/dopex-v2-clamm/pull/1/commits/a0c5a77272a791524a12611e5a28b0c08817f812

4. Gas - Fetch item from the map only once

Storing fetched values from the map can provide gas reduction.

Technical Details

Value from map `ttlToVEID` is fetched twice inside the same function. Storing the value in the variable will reduce gas cost.

Impact

Gas savings.

Recommendation

Fetch the mapping value only once and store it in a variable. Gas savings data from provided tests:

```
testSettleOptionPutITMRoll() (gas: -152 (-0.010%))
testExercisePutOptionRoll() (gas: -152 (-0.010%))
testSettleOptionCallITMRoll() (gas: -152 (-0.010%))
testExerciseCallOptionRoll() (gas: -152 (-0.010%))
testSplitPosition() (gas: -152 (-0.013%))
testSettleOptionPutOTMRoll() (gas: -152 (-0.015%))
testSettleOptionCallOTMRoll() (gas: -152 (-0.015%))
testBuyPutOptionRoll() (gas: -152 (-0.016%))
testBuyCallOptionRoll() (gas: -152 (-0.016%))
Overall gas change: -1368 (-0.006%)
```

Developer Response

Fix - https://github.com/dopex-io/dopex-v2-clamm/pull/1/commits/26993c2a79e4efc1c449b49af6933f5a8c6a4d1f

5. Gas - Use return data instead of fetching it again from the pool

`PositionManager` returns pool token addresses which can be used instead of fetching tokens from pool contract.

Technical Details

Pool contract function is called to get token0 address, but this data is already available for calling position manager. We can verify that handler returns the same address of token0. This value is initialized when creating new storage `tki` to pool token0.

Impact

Gas savings.

Recommendation

Use return token values from position manager:

```
(address[] memory tokens, uint256[] memory amounts, ) = positionManager.usePosition(
    _params.optionTicks[i]._handler,
    usePositionData
);

if (tokens[0] == assetToUse) {
```

Gas savings data from provided tests:

```
testSettleOptionPutITMRoll() (gas: -1019 (-0.069%))
testExercisePutOptionRoll() (gas: -1019 (-0.069%))
testSettleOptionCallITMRoll() (gas: -1019 (-0.070%))
testExerciseCallOptionRoll() (gas: -1019 (-0.070%))
testSplitPosition() (gas: -1019 (-0.086%))
testSettleOptionPutOTMRoll() (gas: -1019 (-0.099%))
testSettleOptionCallOTMRoll() (gas: -1019 (-0.101%))
testBuyPutOptionRoll() (gas: -1019 (-0.107%))
testBuyCallOptionRoll() (gas: -1019 (-0.110%))
Overall gas change: -9171 (-0.037%)
```

Developer Response

Fix - https://github.com/dopex-io/dopex-v2-clamm/pull/1/commits/15519d16db47b7c577ea0fbd063b80e08d89e32e

6. Gas - Fetch item from list only once

If the item from the list is used multiple times, it's cheaper to fetch it only once.

Technical Details

Inside function `mintOptionRoll()` item `_params.optionTicks[i]` is fetched multiple times. It is more gas efficient to fetch it only once inside a loop.

Impact

Gas savings.

Recommendation

Define struct item and fetch it only once inside the first and the second loop.

```
OptionTicks memory opTick;
for (uint256 i; i < _params.optionTicks.length; i++) {
    opTick = _params.optionTicks[i];
```

Gas savings data from provided tests:

`testSettleOptionPutITMRoll() (gas: -4671 (-0.315%))`
`testExercisePutOptionRoll() (gas: -4671 (-0.317%))`
`testSettleOptionCallITMRoll() (gas: -4671 (-0.319%))`
`testExerciseCallOptionRoll() (gas: -4671 (-0.321%))`
`testSplitPosition() (gas: -4671 (-0.395%))`
`testSettleOptionPutOTMRoll() (gas: -4671 (-0.453%))`
`testSettleOptionCallOTMRoll() (gas: -4671 (-0.463%))`
`testBuyPutOptionRoll() (gas: -4671 (-0.491%))`
`testBuyCallOptionRoll() (gas: -4671 (-0.503%))`
`Overall gas change: -42039 (-0.169%)`

Developer Response

Fix - https://github.com/dopex-io/dopex-v2-clamm/pull/1/commits/4db7f1391f0b84d6ab3c55b9ee9aa0938acef51b

7. Gas - Swap deadline value

The swap deadline value can be set to block timestamp because it's executed inside a block to save gas.

Technical Details

The swap deadline value has added 5 days inside the handler and the swapper. Adding more time to the deadline won't have any effect because the swap will always be executed inside a block.

Impact

Gas savings.

Recommendation

For deadline value, use just block timestamp:

```
-deadline: block.timestamp + 5 days
+deadline: block.timestamp
```

Gas savings data from provided tests:

`testUnusePosition() (gas: -54 (-0.003%))`
`testBurnPosition() (gas: -53 (-0.004%))`
`testUsePosition() (gas: -67 (-0.004%))`
`testMintPositionWithSwaps() (gas: -67 (-0.005%))`
`testSettleOptionPutITMRoll() (gas: -175 (-0.012%))`
`testExercisePutOptionRoll() (gas: -175 (-0.012%))`
`testSettleOptionCallITMRoll() (gas: -175 (-0.012%))`

`testExerciseCallOptionRoll() (gas: -175 (-0.012%))`

`Overall gas change: -941 (-0.004%)`

Developer Response

Fix - https://github.com/dopex-io/dopex-v2-clamm/pull/1/commits/9101d6c0fde678144628708bd4e03572f583e96 and https://github.com/dopex-io/dopex-v2-clamm/pull/1/commits/4379bfd0c51b7796d7f81ceac9cbbd70aa46e05e

8. Gas - Set variables as immutable

Set variables as immutable if they cannot be changed to save gas.

Technical Details

`swapRouter` is set only in the constructor and can be changed to immutable. Additional variables in DopexV2OptionPools contract can be set as immutable: `positionManager`, `primePool`, `callAsset` and `putAsset`.

Impact

Gas savings.

Recommendation

Set variables as immutable. Gas savings data from provided tests:

`testPutOptionSim() (gas: -28 (-0.002%))`

`testCallOptionSim() (gas: -28 (-0.002%))`

`testMintPosition() (gas: -28 (-0.004%))`

`testUnusePosition() (gas: -306 (-0.015%))`

`testUsePosition() (gas: -270 (-0.016%))`

`testBurnPosition() (gas: -260 (-0.018%))`

`testMintPositionWithSwaps() (gas: -270 (-0.019%))`

`testDonation() (gas: -1780 (-0.118%))`

`testSplitPosition() (gas: -10361 (-0.876%))`

`testSettleOptionPutITMRoll() (gas: -13502 (-0.909%))`

`testExercisePutOptionRoll() (gas: -13502 (-0.915%))`

`testSettleOptionCallITMRoll() (gas: -13629 (-0.931%))`

`testExerciseCallOptionRoll() (gas: -13629 (-0.936%))`

`testSettleOptionCallOTMRoll() (gas: -10921 (-1.083%))`

`testBuyCallOptionRoll() (gas: -10361 (-1.116%))`

`testSettleOptionPutOTMRoll() (gas: -12794 (-1.241%))`

`testBuyPutOptionRoll() (gas: -12234 (-1.287%))`

`Overall gas change: -113903 (-0.459%)`

Developer Response

Fix - https://github.com/dopex-io/dopex-v2-clamm/pull/1/commits/9bd21531934505a14ebe810aaaf1375b75bf3d3b and https://github.com/dopex-io/dopex-v2-clamm/pull/1/commits/cc69ef4507e3157ab20b59cae25d5301e6a55006

9. Gas - Remove duplicated code

Removing duplicated code will lower the gas cost and the code will be simpler.

Technical Details

Calling function `_feeCalculation()` is called in both inside if and else statement. Using just one call before if statement will reduce code and gas cost.

Impact

Gas savings.

Recommendation

Implementing the suggested solution with only one `_feeCalculation()` will reduce gas cost:

`testBurnPosition() (gas: -4 (-0.000%))`

`testUsePosition() (gas: -6 (-0.000%))`

`testPutOptionSim() (gas: -6 (-0.000%))`

`testCallOptionSim() (gas: -6 (-0.000%))`

`testMintPositionWithSwaps() (gas: -6 (-0.000%))`

`testUnusePosition() (gas: -10 (-0.000%))`

`testDonation() (gas: -9 (-0.001%))`

`testMintPosition() (gas: -6 (-0.001%))`

`Overall gas change: -53 (-0.000%)`

Developer Response

Fix - https://github.com/dopex-io/dopex-v2-clamm/pull/1/commits/6fb1faab1218356f6a9875158b5d3a7050cfbb72

10. Gas - Duplicated reentrancy

Reentrancy checks are duplicated in `UniswapV3SingleTickLiquidityHandler` which results in higher gas costs for some calls.

Technical Details

Handler external function can be called only by whitelisted apps, in this case, `DopexV2PositionManager`. Functions `mintPositionHandler()` and `burnPositionHandler()` have reentrancy protection but these function should be called only from `DopexV2PositionManager` functions `mintPosition()` and `burnPosition()` which also have reentrancy protection.

Impact

Gas savings.

Recommendation

With the current system architecture, where DopexV2PositionManager is the only entry point for all user interactions, reentrancy protection can be handled only in the position manager. If all calls to the handler are going through the whitelisted app, position manager, it makes sense to move all reentrancy protection to position manager only. `ReentrancyGuard` can be removed from the handler, but additional calls in the position manager must have reentrancy protection like `usePosition()`, `unusePosition()` and `donateToPosition()`. Gas savings data from provided tests:

`testSettleOptionPutITMRoll() (gas: -4 (-0.000%))`

`testExercisePutOptionRoll() (gas: -4 (-0.000%))`

`testSettleOptionCallITMRoll() (gas: -4 (-0.000%))`

`testExerciseCallOptionRoll() (gas: -4 (-0.000%))`

`testSettleOptionPutOTMRoll() (gas: -4 (-0.000%))`

`testSettleOptionCallOTMRoll() (gas: -4 (-0.000%))`

`testSplitPosition() (gas: -5 (-0.000%))`

`testBuyPutOptionRoll() (gas: -5 (-0.001%))`

`testBuyCallOptionRoll() (gas: -5 (-0.001%))`

`testUsePosition() (gas: -2748 (-0.164%))`

`testPutOptionSim() (gas: -2750 (-0.194%))`

`testCallOptionSim() (gas: -2750 (-0.194%))`

`testMintPositionWithSwaps() (gas: -2745 (-0.197%))`

`testMintPosition() (gas: -2745 (-0.407%))`

`testBurnPosition() (gas: -11752 (-0.796%))`

`testUnusePosition() (gas: -16833 (-0.799%))`

`testDonation() (gas: -21909 (-1.450%))`

`Overall gas change: -64271 (-0.259%)`

This is a recommendation for a given protocol design with only one position manager. If there is a plan to add more position managers, be sure to add reentrancy checks. Also, if the other contracts are whitelisted to call the handler, those contracts must be taken into account for reentrancy checks.

Developer Response

Fix - https://github.com/dopex-io/dopex-v2-clamm/pull/1/commits/f0ba59344f12926f9682e8ee11ca47cfda8aac72

11. Gas - Remove constant if it is not used

Removing the unused constant can lower the gas cost.

Technical Details

Constant `DEFAULT_LOCKED_BLOCK_DURATION` is not used in the code, except just to assign default value which can be done more efficiently with constant variable.

Impact

Gas saving.

Recommendation

Remove constant `DEFAULT_LOCKED_BLOCK_DURATION` and assign its value directly to `lockedBlockDuration`. Gas savings data from provided tests:

```
testSplitPosition() (gas: -66 (-0.006%))
testBuyPutOptionRoll() (gas: -66 (-0.007%))
testBuyCallOptionRoll() (gas: -66 (-0.007%))
testSettleOptionPutITMRoll() (gas: -110 (-0.007%))
testExercisePutOptionRoll() (gas: -110 (-0.007%))
testSettleOptionCallITMRoll() (gas: -110 (-0.008%))
testExerciseCallOptionRoll() (gas: -110 (-0.008%))
testBurnPosition() (gas: -140 (-0.009%))
testSettleOptionPutOTMRoll() (gas: -110 (-0.011%))
testSettleOptionCallOTMRoll() (gas: -110 (-0.011%))
testUsePosition() (gas: -198 (-0.012%))
testMintPositionWithSwaps() (gas: -176 (-0.013%))
testUnusePosition() (gas: -299 (-0.014%))
testPutOptionSim() (gas: -220 (-0.016%))
testCallOptionSim() (gas: -220 (-0.016%))
testDonation() (gas: -246 (-0.016%))
testMintPosition() (gas: -132 (-0.020%))
Overall gas change: -2489 (-0.010%)
```

Developer Response

Fix - https://github.com/dopex-io/dopex-v2-clamm/pull/1/commits/3a0b1884c1fae20b6d1db624927bf3a438d621bd

12. Gas - Function parameter can be removed when it's constant

Function parameter that is always the same can be removed as a parameter to save gas.

Technical Details

Functions `_convertToShares()` and `_convertToAssets()` are called always called with the same value.

Impact

Gas savings.

Recommendation

Remove param `rounding` from both functions. `_convertToShares()` uses `Math.Rounding.Down` and `_convertToAssets()` uses `Math.Rounding.Up` as a constant value. Gas savings data from provided tests:

```
testUsePosition() (gas: -5 (-0.000%))
testPutOptionSim() (gas: -5 (-0.000%))
testCallOptionSim() (gas: -5 (-0.000%))
testMintPositionWithSwaps() (gas: -5 (-0.000%))
testMintPosition() (gas: -5 (-0.001%))
testBurnPosition() (gas: -12 (-0.001%))
testUnusePosition() (gas: -20 (-0.001%))
testDonation() (gas: -28 (-0.002%))
Overall gas change: -85 (-0.000%)
```

Developer Response

Fix - https://github.com/dopex-io/dopex-v2-clamm/pull/1/commits/2b55f0de911e54f8cbd5f0bf6bc6d2db344497789

Informational Findings

1. Informational - Incompatible with fee-on-transfer tokens

The transfer logic in the system will fail for fee-on-transfer tokens because of transfers through PositionManager. There currently is no plan to use this code with fee-on-transfer tokens, but this limitation should be documented in case there are strategy changes in the future.

Technical Details

In position manager, `mintPosition()` has a combination of transfer from the user to the position manager and approval of the same amount to the handler. The handler will try to transfer the same amount which will revert when transferring a fee-on-transfer token. This is because the initial transfer from the user will result in less funds for the handler transfer call, but it is expecting the same amount.

Impact

Informational.

Recommendation

If there are plans to support other tokens in the future, rethink about DopexV2PositionManager contract role in the system. Now, it is used as a router to transfer calls and tokens which will need additional calculations to support fee-on transfer tokens.

Developer Response

There is no plan to support fee-on-transfer tokens at the moment.

However, we can also create a special handler to pass in additional data to the PositionManager or directly take the tokens via Handler itself.

2. Informational - Use indexed values in events

The indexed parameters for logged events will allow you to search for these events using the indexed parameters as filters.

Technical Details

Event values can be indexed to enable easier searching.

- In DopexV2OptionPools add the attribute `indexed` to `address user` and `uint256 tokenId` in all events.
- In UniswapV3SingleTickLiquidityHandler add the attribute `indexed` to `address user` and `uint256 tokenId` in multiple events.

Impact

Informational.

Recommendation

Indexing key values in events can help in better off-chain analysis.

Developer Response

Acknowledged

3. Informational - Using spot price

The premium amount for creating options is calculated using the pool spot price which can be manipulated.

Technical Details

`premiumAmount` value is calculated using the prime pool spot price. Spot price can be manipulated to pay a lower premium amount which would harm users that provided liquidity. Good design is to define `primePool` instead of using a user-provided pool for spot price.

Contract owners have the option to change the option pricing contract which can change premium calculation if needed. Option pricing contracts were not in the scope of this audit, so the math behind price manipulation for premium amounts cannot be verified.

Impact

Informational.

Recommendation

Carefully pick which pools can be created and which pools will be picked for prime pool and set up volatility parameters that are in line with the market. Small liquidity pools are easy and cheap to manipulate the spot price which can lead to paying a much lower premium.

Set up monitoring for prime pool liquidity and adjust the volatility parameter to market conditions. Also, monitor CLAMM liquidity because high liquidity for creating options could enable attackers to pay a high price for manipulating the stop price. Premium amount is multiple with the

amount used for creating the option. This means that high liquidity on CLAMM for specific ticks can justify paying for price manipulation if the final premium amount can be lower for more than the price paid for manipulation.

Developer Response

The pools will be picked up by the admin, making sure that it has decent liquidity.

4. 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

DopexV2OptionPools updates the storage variable but doesn't emit events. Functions that could have events added include:

- `updateExerciseDelegate()`
- `updateIVs()`
- `updateAddress()` - this function could be broken down into few functions. This way it will be easier to track new values and will be safer to call update functions. Now, there is a possibility of unintentionally setting values to zero.

Impact

Informational.

Recommendation

Add events to the functions listed above.

Developer Response

Acknowledged

5. Informational - Options token id

DopexV2OptionPools token id is initialized to 1 and the first token id will have value 2.

Technical Details

The token id counter is initialized to 1, but before each minting counter is incremented. This means that the first token id will be 2.

Impact

Informational.

Recommendation

Skip initializing token id to 1 to set the first token to value 1 as expected and to save gas.

```
- uint256 public optionIds = 1;
+ uint256 public optionIds;
```

Developer Response

Fix - https://github.com/dopex-io/dopex-v2-clamm/pull/9

6. Informational - Last share cannot be withdrawn

Withdrawing all shares from UniswapV3SingleTickLiquidityHandler will revert if the user holds all shares.

Technical Details

Converting shares to assets is done by rounding up the final number of assets. This means that the withdrawal or burning of all possible shares will revert. The user can withdraw all shares, and liquidity, but the last share must be left because of rounding up.

For example, a user deposits 100 assets and gets 100 shares in return. When he tries to withdraw 100 share, it is converted to 101 assets.

PoC for PositionManagerHandlerTest that demonstrates the user cannot withdraw all shares:

```
    function testMintAndBurnPosition() public {
        uint256 amount0 = 0;
```

```
        uint256 amount1 = 5e18;

        int24 tickLower = -77520; // 2299.8
        int24 tickUpper = -77510; // 2302.1

        uint256 shares = positionManagerHandler.mintPosition(
            token0,
            token1,
            amount0,
            amount1,
            tickLower,
            tickUpper,
            pool,
            bob
        );

        uint256 bobBalance = uniV3Handler.balanceOf(
            bob,
            positionManagerHandler.getTokenId(pool, tickLower, tickUpper)
        );
        assertEq(bobBalance, shares);

        positionManagerHandler.burnPosition(
            bobBalance,
            tickLower,
            tickUpper,
            pool,
            bob
        );

        bobBalance = uniV3Handler.balanceOf(
            bob,
            positionManagerHandler.getTokenId(pool, tickLower, tickUpper)
        );
        assertEq(bobBalance, 0);
    }
```

Impact

Informational. The user won't lose funds but will keep the last share that cannot be withdrawn. This will break user flow when withdrawing all shares.

Recommendation

One possibility is when minting the first shares, remove one share to compensate for rounding up of last share.

```
sharesMinted = liquidity - 1;
```

This could produce a small loss for the first depositor, but the final withdrawer can withdraw all shares. Another option is to limit the last withdrawal on the frontend to the number of shares minus 1 share, so the transaction won't revert. The most important thing is to limit the possibility of reverting when the last user tries to withdraw all shares.

Developer Response

Yes, the front-end already subtracts 1 share, if the withdrawal is from the last depositor.

7. Informational - Upgrade OpenZeppelin dependency

The OpenZeppelin contracts dependency is version 4.9.0, which is outdated. Consider updating to a newer version.

Technical Details

OZ library v4.9.0 is not the latest version available. Consider upgrading to v4.9.3 which fixes some minor issues in v4.9.0.

Impact

Informational.

Recommendation

Upgrade OZ dependency to a newer version.

Developer Response

Acknowledged.

8. Informational - High centralization risk throughout protocol

DopexV2 CLAMM contracts are architected in a way that involves a high level of centralization risk. Centralization can offer many benefits like faster response times from governance to adapt the protocol, but also downsides. Users interacting with the protocol should be aware of the risks involved with this design choice because rogue governance can withdraw protocol value to arbitrary addresses.

Technical Details

Every DeFi protocol must decide how immutable or centralized the protocol will be. Dopex's design falls on the very centralized end of the spectrum. This is by design and does not add risks if governance actions are only performed by trusted parties. However, contract owners have the ability to take the funds of the users or lock them in multiple ways.

1. `DopexV2OptionPools` holds the funds that are used for creating options and the contract owner have an option to withdraw the funds anytime.
2. `UniswapV3SingleTickLiquidityHandler` holds the liquidity but it has an option for the contract owner to withdraw liquidity and withdraw user tokens from the contract. Function `forceWithdrawUniswapV3Liquidity()` will remove all liquidity for a given pool and ticks. This is done without updating the internal account for position which can lead to irreversible system state.
3. The functions that enable users to exit the protocol, like `burnPositionHandler()` can be paused which can lead to locked user funds. Pausing functionality is limited to the contract owner.
4. User enter and exit position using `DopexV2PositionManager` which has a mapping of whitelisted handlers. The contract owner can unlist the handler in which the user has deposited funds and lock the user because the user cannot withdraw directly from the handler but must use position manager.

In summary, the centralized nature of DopexV2 CLAMM adds some risk because of the power that governance has. This finding has a risk of Informational only because the assumption was made that the protocol has trustworthy governance. If this assumption does not hold, the risk severity would likely jump to Critical. Even if all governance operates as a trusted party, centralization brings with it the risk of private key theft which has impacted several DAOs in the last couple of years.

Impact

Informational.

Recommendation

The centralized design choices have likely been made for good reason, but it would help to document very clearly what safety measures are taken around the multisig and governance models that will be involved in V2 to minimize the risk of malicious actions (intentional or not).

Developer Response

The Dopex team monitors contracts and executes admin functionality (behind the team multisig) depending on requirements.

9. Informational - Misleading NatSpec description

NatSpec comment should provide the correct description of the function is doing.

Technical Details

The function `positionSplitter()` provides the notice: "Splits the given option roll into two new option rolls" but the implementation is different. The provided option is split between only one new option and the provided one.

Impact

Informational.

Recommendation

Be correct in NatSpec comments.

Developer Response

Fix - https://github.com/dopex-io/dopex-v2-clamm/pull/1/commits/03916fecc174c7f7691a431e3bdbc2c778f5205b

10. Informational - Inconsistency in encoding and decoding integers

There is inconsistency in encoding and decoding `uint`, integer sizes.

Technical Details

When encoding `UsePositionData` in DopexV2OptionPools last value liquidity is in uint256 while in decoding its expecting uint128. The same problem is applicable for `UnusePositionParams` and `DonateParams`. Here encoded donate amount is uint256 but decoding it in uint128. Calling unuse position is also encoding uint256 value.

Impact

Informational.

Recommendation

Use the same size integers for encoding and decoding. Be aware that decoding would fail if the value above the uint128 maximum value is sent.

Developer Response

Since the liquidity will be always below uint128, I don't see this as an issue.

11. Informational - Remove code that is not used

Some code can be removed without changing the functionalities to improve code readability.

Technical Details

`else` part of the statement is defined even though doesn't have any functionality.

Impact

Informational.

Recommendation

Remove code that is not needed.

Developer Response

I want to keep the else condition empty, as I have further logic in the future which could be added here to make the Option Market much better.

12. Informational - Validated values as early as possible

Validating values at the start of the function will provide clear code and potentially lower gas costs for failed transactions.

Technical Details

- Here is the validation for `amount0` and `amount1` inside a function `_compoundFees()`, but this validation should be done immediately after the values are calculated inside `mintPositionHandler()`.
- In `DopexV2OptionPools`, validation that the pool is approved can be done before making state changes to `opTickMap`.
- In function `positionSplitter()` storage variable `optionIds` should be increased only after the first and the second validation has passed.

Impact

Informational.

Recommendation

Validate values as early as possible.

Developer Response

Fix - https://github.com/dopex-io/dopex-v2-clamm/blob/feat/events/src/handlers/UniswapV3SingleTickLiquidityHandler.sol#L196

13. Informational - Non-descriptive variable names

Some variable names do not clearly describe what the variable values refer to.

Technical Details

Some variable names could be improved:

- `a00` to `userLiquidity0`.
- `a11` to `userLiquidity1`.
- `a0` to `totalLiquidity0`.
- `a1` to `totalLiquidity1`.

Impact

Informational.

Recommendation

Give variables more accurate names to avoid confusion.

Developer Response

Fix - https://github.com/dopex-io/dopex-v2-clamm/pull/1/commits/e84fecb5abfed698590e406686171896aa41740c

Final remarks

DopexV2 CLAMM is a novel approach to providing an on-chain options market by using Uniswap V3 concentrated liquidity positions as collateral to write options. The focus of the system is to connect users providing liquidity for narrow liquidity ranges with traders who want to create options. If the Uniswap V3 liquidity position is used by an options buyer, the Uniswap V3 liquidity position provider gets paid a premium which will be higher than the earned fees from Uniswap V3 alone. Calculating the premium is done using a contract that wasn't in the scope of this review. If the Uniswap V3 liquidity position is not used by an options buyer, it can earn the normal fees from Uniswap V3. These fees are auto-compounded and reinvested in the position with no slippage protection, meaning that MEV bots may be able to extract value from the earned fees.

The system design does not use a decentralized price oracle like Chainlink, meaning that the system is vulnerable to price manipulation. The system is also vulnerable to MEV because of the lack of slippage protection. On the other hand, the choice not to rely on a decentralized oracle provider allows the protocol to support a greater range of tokens, because any token that has a Uniswap V3 pool can be used in the protocol.

Building on top of Uniswap V3 allows users to choose the range of liquidity they want to provide. Picking a wide liquidity range can lead to locked liquidity and low returns. This should be prevented with proper logic checks in the frontend.