Cron Finance Security Review

Auditors

Kurt Barry, Lead Security Researcher

Noah Marconi, Lead Security Researcher

M4rio.eth, Security Researcher

Calvin Boehr, Junior Security Researcher

Christos Papakonstantinou, Junior Security Researcher

Report prepared by: Pablo Misirov

April 11, 2023

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Novel on-chain trade execution engine called TWAMM for permissionless, transparent & gas efficient large token swaps.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of TWAMM according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 13 days in total, Cron Finance engaged with Spearbit to review the TWAMM protocol. In this period of time a total of **43** issues were found.

Note: See appendix for more information regarding the extension period.

Summary

Project Name	Cron Finance
Repository	twamm
Commit	d061e0...f514
Type of Project	AMM, DeFi
Audit Timeline	Jan 23 - Feb 8
Two week fix period	Feb 8 - Feb 22
Extension period	April 5th - April 7th

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	5	5	0
Medium Risk	0	0	0
Low Risk	14	12	2
Gas Optimizations	9	9	0
Informational	15	12	3
Total	**43**	**38**	**5**

5 Findings

5.1 High Risk

5.1.1 Balancer Read-Only Reentrancy Vulnerability (Changes from dev team added to audit.)

Severity: High Risk

Context: CronV1Pool.sol#L1250

Description: Balancer's read-only reentrancy vulnerability potentially effects the following Cron-Fi TWAMM functions:

- getVirtualReserves
- getVirtualPriceOracle
- executeVirtualOrdersToBlock A mitigation was provided by the Balancer team that uses a minimum amount of gas to trigger a reentrancy check. The Balancer vulnerability is discussed in greater detail here:
- reentrancy-vulnerability-scope-expanded/4345

Recommendation: Install the mitigation into the aforementioned methods, changing them to non-view functions, but documenting that they do not meaningfully modify state. If possible, confirm that the mitigation is not needed by testing the methods without it and removing it if shown to not be a problem.

Twamm: Addressed in commit 5a529da.

Spearbit: Verified.

5.1.2 Overpayment of one side of LP Pair onJoinPool due to sandwich or user error

Severity: High Risk

Context: CronV1Pool.sol#L2048-L2051

Description: Only one of the two incoming tokens are used to determine the amount of pool tokens minted (amountLP) on join

```
amountLP = Math.min(
    _token0InU112.mul(supplyLP).divDown(_token0ReserveU112),
    _token1InU112.mul(supplyLP).divDown(_token1ReserveU112)
);
```

In the event the price moves between the time a minter sends their transaction and when it is included in a block, they may overpay for one of _token0InU112 or _token1InU112. This can occur due to user error, or due to being sandwiched.

Concrete example:

```
pragma solidity ^0.7.0;

pragma experimental ABIEncoderV2;

import "forge-std/Test.sol";

import "../HelperContract.sol";
import { C } from "../../Constants.sol";
import { ExecVirtualOrdersMem } from "../../Structs.sol";

contract JoinSandwich is HelperContract {

  uint256 WAD = 10**18;

  function testManualJoinSandwich() public {
    address userA = address(this);
    address userB = vm.addr(1323);

    // Add some base liquidity from the future attacker.
    addLiquidity(pool, userA, userA, 10**7 * WAD, 10**7 * WAD, 0);
    assertEq(CronV1Pool(pool).balanceOf(userA), 10**7 * WAD - C.MINIMUM_LIQUIDITY);

    // Give userB some tokens to LP with.
    token0.transfer(userB, 1_000_000 * WAD);
    token1.transfer(userB, 1_000_000 * WAD);
    addLiquidity(pool, userB, userB, 10**6 * WAD, 10**6 * WAD, 0);
    assertEq(CronV1Pool(pool).balanceOf(userB), 10**6 * WAD);
    exit(10**6 * WAD, ICronV1Pool.ExitType(0), pool, userB);
    assertEq(CronV1Pool(pool).balanceOf(userB), 0);

    // Full amounts are returned b/c the exit penalty has been removed (as is being done anyway).
    assertEq(token0.balanceOf(userB), 1_000_000 * WAD);
    assertEq(token1.balanceOf(userB), 1_000_000 * WAD);

    // Now we'll do the same thing, simulating a sandwich from userA.
    uint256 swapProceeds =
      swapPoolAddr(5 * 10**6 * WAD, /* unused */ 0, ICronV1Pool.SwapType(0), address(token0), pool,
↪   userA);

    // Original tx from userB is sandwiched now...
    addLiquidity(pool, userB, userB, 10**6 * WAD, 10**6 * WAD, 0);

    // Sell back what was gained from the first swap.
    swapProceeds =
      swapPoolAddr(swapProceeds, /* unused */ 0, ICronV1Pool.SwapType(0), address(token1), pool, userA);
    emit log_named_uint("swapProceeds 1 to 0", swapProceeds);  // allows seeing what userA lost to fees

    // Let's see what poor userB gets back of their million token0 and million token1...
    assertEq(token0.balanceOf(userB), 0);
    assertEq(token1.balanceOf(userB), 0);
    exit(ICronV1Pool(pool).balanceOf(userB), ICronV1Pool.ExitType(0), pool, userB);
    emit log_named_uint("userB token0 after", token0.balanceOf(userB));
    emit log_named_uint("userB token1 after", token1.balanceOf(userB));
  }
}
```

Output:

```
Logs:
  swapProceeds 1 to 0: 4845178856516554015932796
  userB token0 after: 697176321467715374004199
  userB token1 after: 687499999999999999999999
```

1. We have a pool where the attacker is all of the liquidity (10^7 of each token)
2. A LP tries to deposit another 10^6 in equal proportions
3. The attacker uses a swap of $5 * 10^6$ of one of the tokens to distort the pool. They lose about 155k in the process, but the LP loses far more, nearly all of which goes to the attacker--about 615,324 (sum of the losses of the two tokens since they're equally priced in this example).

The attacker could be a significantly smaller proportion of the pool and still find this attack profitable. They could also JIT the liquidity since the early withdrawal penalty has been removed.

The attack becomes infeasible for very large pools (has to happen over multiple TXs so can't flash loan --need own capital), but is relevant in practice.

Recommendation: Allow LPs to specify slippage tolerance on both join and exit. An example of this is the uniswap/v2-periphery/UniswapV2Router02.sol where slippage protection is added both when adding and removing liquidity.

Twamm: Addressed in commit `e0e5fb4`.

Followed recommendation for `onJoin`, adding minimum price arguments to allow users to specify asymmetric limits if desired. However, for the exit use case (i.e. an exit price attack), the existing Balancer scaffolding wrapping our pool provides a minimum exit `amounts` that `reverts` if at least a specified number of tokens is received.

To ensure this works with our pool I added a test loosely based on yours with a non-sensical price attack to illustrate that users can reject liquidation during undesirable price movements (see contracts/twault/test/auto/LiquidityAttackAuto.t.sol function `testAutoExitProtectedAgainstSandwichMovement`, which manipulates price, specifies minimums and is reverted with BAL#505).

Spearbit: Confirmed.

5.1.3 Loss of Long-Term Swap Proceeds Likely in Pools With Decimal or Price Imbalances

Severity: High Risk

Context: VirtualOrders.sol#L166

Description: This TWAMM implementation tracks the proceeds of long-term swaps efficiently via accumulated values called "scaled proceeds" for each token. In every order block interval (OBI), the scaled proceeds for e.g. the sale of token 0 are incremented by

`(quantity of token 1 purchased during the OBI)` $*2^{6}4/$ `(sales rate of token 0 during the OBI)`

Then the proceeds of any specific long-term swap can be computed as the product of the difference between the scaled proceeds at the current block (or the expiration block of the order if filled) and the last block for which proceeds were claimed for the order and the order's sales rate, divided by $2^{6}4$:

`last` := $min(currentBlock, orderExpiryBlock)$

`prev` := block of last proceeds collection, or block order was placed in if this is the first withdrawal

`LT swap proceeds` = $(scaledProceeds_{l}ast - scaledProceeds_{p}rev) * (ordersalesrate)/2^{6}4$

The value $2^{6}4$ is referred to as the "scaling factor" and is intended to reduce precision loss in the division to determine the increment to the scaled proceeds.

The addition to increment the scaled proceeds and the subtraction to compute its net change is both intentionally done with unchecked arithmetic--since only the difference matters, so long as at most one overflow occurs between claim-of-proceeds events for any given order, the computed proceeds will be correct (up to rounding errors). If two or more overflows occur, however, funds will be lost by the swapper (unclaimable and locked in the contract).

Additionally, to cut down on gas costs, the scaled proceeds for the two tokens are packed into a single storage slot, so that only 128 bits are available for each value. This makes multiple overflows within the lifetime of a single order more likely. The CronFi team was aware of this at the start of the audit and specifically requested it be investigated, though they expected a maximum order length of 5 years to be sufficient to avoid the issue in practice.

The scaling factor of $2^{6}4$ is approximately $1.8 * 10^{1}9$, close to the unit size of an 18-decimal token. It indeed works well if both pool tokens have similar decimals and relative prices that do not differ by too many orders of magnitude, as the quantity purchased and the sales rate will then be of similar magnitude, canceling to within a few powers of ten ($2^{1}28$ $3.4 * 10^{3}8$, leaving around 19 orders of magnitude after accounting for the scaling factor). However, in pools with large disparities in price, decimals, or both, numerical issues are easy to encounter.

The most extreme, realistic example would be a DAI-GUSD pool. DAI has 18 decimals while GUSD has only 2. We will treat the price of DAI and GUSD as equal for this analysis, as they are both stablecoins, and arbitrage of the TWAMM pool should prevent large deviations. Selling GUSD at a rate of 1000 per block, with an OBI of 64 (the stable pool order block interval in the audited commit) results in an increment of the scaled proceeds per OBI of:

$increment = (64 * 1000 * 10^{1}8) * 2^{6}4/(1000 * 10^{2}) = 1.18 * 10^{3}7$

This will overflow an unsigned 128 bit integer after 29 OBIs; at 12 seconds per block, this means the first overflow occurs after $12 * 64 * 29 = 22272$ seconds or about 6.2 hours, and thus the first double overflow (and hence irrevocable loss of proceeds if a withdrawal is not executed in time) will occur within about 12.4 hours (slightly but not meaningfully longer if the price is pushed a bit below 1:1, assuming a deep enough pool or reasonably efficient arbitrage). Since the TWAMM is intended to support swaps that take days, weeks, months, or even years to fill, without requiring constant vigilance from every long-term swapper, this is a strong violation of safety.

A less extreme but more market-relevant example would be a DAI-WBTC pool. WBTC has 8 instead of 2 decimals, but it is also more than four orders of magnitude more valuable per token than DAI, making it only about 2 orders of magnitude "safer" than a DAI-GUSD pool. Imitating the above calculation with 20_000 DAI = 1 WBTC and selling 0.0025 WBTC (~$50) per block with a 257 block OBI yields:

increment = $(257 * 50 * 10^{1}8) * 2^{6}4/(0.0025 * 10^{8}) = 9.48 * 10^{3}5$

OBI to overflow = $\mathit{ceiling}(2^{1}28/(9.48 * 10^{3}5)) = 359$

time to overflow = $12 * 257 * 359$ = 1107156 seconds = 307 hours = 12.8 days

, or a little more than a day to encounter the second overflow. While less bad than the DAI-GUSD example, this is still likely of significant concern given that the CronFi team indicated these are parameters under which the TWAMM should be able to function safely and DAI-WBTC is a pair of interest for the v1 product. It is worth noting that these calculations are not directly dependent on the quantity being sold so long as the price stays roughly constant--any change in the selling rate will be compensated by a proportional change in the proceeds quantity as their ratio is determined by price. Thus the analysis depends only on relative price and relative decimals, to a good approximation--so a WBTC-DAI pool can be expected to experience an overflow roughly every two weeks at prevailing market prices, so long as the net selling rate is non-zero.

Recommendation: Consider adjusting the scaling factor based on the relative decimals of the two tokens (this will result in two different scaling factors per pool). The lower the selling rate in absolute terms, the smaller the factor needed to avert serious precision loss. This is similar in effect to scaling all balances to, say, 18 decimals but should be simpler to implement. Highly imbalanced prices are still a risk, but these are unlikely to be as extreme as the differences that arise from unequal decimals. The scaling factor could be made configurable upon pool creation to allow an adjustment for price as well as decimals, although if prices diverge significantly after creation, there is likely no other option than abandoning a pool (as retroactively correcting scaled proceeds would be cost-prohibitive). Alternatively, separate 256-bit containers can be used to store scaled proceeds, although this will greatly increase the gas costs of virtual order execution.

Twamm: Addressed in the following commits: faf1d5e, bfb6a5c, 162461a.

Spearbit: Verified.

5.1.4 An attacker can block any address from joining the Pool and minting BLP Tokens by filling the joinEventMap mapping.

Severity: High Risk

Context: JoinEventLib.sol#L65-L137, Constants.sol#L112

Description: An attacker can block any address from minting *BLP Tokens*. This occurs due to the MAX_JOIN_EVENTS limit, which is present in the JoinEventLib library. The goal for an attacker is to block a legitimate user from minting *BLP Tokens*, by filling the joinEventMap mapping.

The attacker can fill the joinEventMap mapping by performing the following steps:

- The attacker mints *BLP Tokens* from 50 different addresses.
- Each address transfers the *BLP Tokens*, alongside the join events, to the user targeted with a call to the CronV1Pool(pool).transfer and CronV1Pool(pool).transferJoinEvent functions respectively. Those transfers should happen in different blocks.

After 50 blocks (50 * 12s = 10 minutes) the attacker has blocked the legitimate user from minting _BLP Tokens_, as the maximum size of the joinEventMap‘ mapping has been reached.

The impact of this vulnerability can be significant, particularly for smart contracts that allow users to earn yield by providing liquidity in third-party protocols. For example, if a governance proposal is initiated to generate yield by providing liquidity in a CronV1Pool pool, the attacker could prevent the third-party protocol from integrating with the CronV1Pool protocol.

A proof-of-concept exploit demonstrating this vulnerability can be found below:

```
function testGriefingAttack() public {
    console.log("-----------------------------");
    console.log("Many Users mint BLP tokens and transfer the join events to the user 111 in order to
↪   fill the array!");

    for (uint j = 1; j < 51; j++) {
      _addLiquidity(pool, address(j), address(j), 2_000, 2_000, 0);
      vm.warp(block.timestamp + 12);

      vm.startPrank(address(j));
      //transfer the tokens
      CronV1Pool(pool).transfer(address(111), CronV1Pool(pool).balanceOf(address(j)));

      //transfer the join events to the address(111)
      CronV1Pool(pool).transferJoinEvent(address(111), 0 , CronV1Pool(pool).balanceOf(address(j)));
      vm.stopPrank();
    }

     console.log("Balance of address(111) before minting LP Tokens himself",
↪   ICronV1Pool(pool).balanceOf(address(111)));

    //user(111) wants to enter the pool
    _addLiquidity(pool, address(111), address(111), 5_000, 5_000, 0);

    console.log("Join Events of user address(111): ",
↪   ICronV1Pool(pool).getJoinEvents(address(111)).length);
    console.log("Balance of address(111) after adding the liquidity: ",
↪   ICronV1Pool(pool).balanceOf(address(111)));
  }
```

Recommendation: A possible mitigation could be to override the _move function of the BalancerPoolToken contract by also transferring the joinEvents alongside the BLP tokens.

Twamm: Join Event / Holding Period & Penalty complete removed in commit 3130e41.

Spearbit: Verified.

5.1.5 The executeVirtualOrdersToBlock function updates the oracle with the wrong block.number

Severity: High Risk

Context: CronV1Pool.sol#L1130

Description: The executeVirtualOrdersToBlock is external, meaning anyone can call this function to execute virtual orders.

The _maxBlock parameter can be lower block.number which will make the oracle malfunction as the oracle update function _updateOracle uses the block.timestamp and assumes that the update was called with the reserves at the current block.

This will make the oracle update with an incorrect value when _maxBlock can be lower than block.number.

Recommendation: Consider adding the block number as a parameter within the _updateOracle in order for this function to not rely on the current block.

Twamm: Addressed in commit f324de7.

Spearbit: Verified.

5.2 Low Risk

5.2.1 The `_join` function does not check if the recipient is address(0)

Severity: Low Risk

Context: CronV1Pool.sol#L2055

Description: As stated within the Balancer's `PoolBalances.sol`

```
// The Vault ignores the `recipient` in joins and the `sender` in exits: it is up to the Pool to keep
↪  track of
// their participation.
```

The recipient is not checked if it's the `address(0)`, that should happen within the pool implementation. Within the Cron implementation, this check is missing which can cause losses of LPs if the recipient is sent as `address(0)`.

This can have a high impact if a 3rd party integration happens with the Cron pool and the "joiner" is mistakenly sending an `address(0)`. This becomes more dangerous if the 3rd party is a smart contract implementation that connects with the Cron pool, as the default value for an address is the `address(0)`, so the probability of this issue occurring increases.

Recommendation: Add a check for the recipient not to be `address(0)`.

Twamm: Addressed in commit `370a0d3a`.

Spearbit: Verified.

5.2.2 Canonical token pairs can be griefed by deploying new pools with malicious admins

Severity: Low Risk

Context: CronV1PoolFactory.sol#L63

Description:

```
function create(
    address _token0,
    address _token1,
    string memory _name,
    string memory _symbol,
    uint256 _poolType,
    address _pauser
  ) external returns (address) {
    CronV1Pool.PoolType poolType = CronV1Pool.PoolType(_poolType);
    requireErrCode(_token0 != _token1, CronErrors.IDENTICAL_TOKEN_ADDRESSES);
    (address token0, address token1) = _token0 < _token1 ? (_token0, _token1) : (_token1, _token0);
    requireErrCode(token0 != address(0), CronErrors.ZERO_TOKEN_ADDRESSES);
    requireErrCode(getPool[token0][token1][_poolType] == address(0), CronErrors.EXISTING_POOL);
    address pool = address(
      new CronV1Pool(IERC20(_token0), IERC20(_token1), getVault(), _name, _symbol, poolType,
↪   address(this), _pauser)
    );
//...
```

Anyone can permissionlessly deploy a pool, with it then becoming the canonical pool for that pair of tokens. An attacker is able to pass a malicious `_pauser` the twamm pool, preventing the creation of a legitimate pool of the same type and tokens. This results in race conditions between altruistic and malicious pool deployers to set the admin for every token pair.

Malicious actors may grief the protocol by attempting to deploy token pairs with and exploiting the admin address, either deploying the pool in a paused state, effectively disabling trading for long-term swaps with the pool, pausing the pool at an unknown point in the future, setting fee and holding penalty parameters to inappropriate values, or setting illegitimate arbitrage partners and lists.

This requires the factory owner to remove the admin of each pool individually and to set a new admin address, fee parameters, holding periods, pause state, and arbitrage partners in order to recover each pool to a usable condition if the griefing is successful.

Recommendation: Remove the user defined `_pauser` parameter from pool creation and pass a default admin address to every pool on creation. Keep in mind that this default admin address is a singular point of centralization for the protocol. Consider deploying a Cron Finance governance address.

Twamm: Addressed in commit `987f539`. Created a 2/3 multisig safe that is the factory owner and default pool admin.

Spearbit: Verified.

5.2.3 Refund Computation in _withdrawLongTermSwap Contains A Risky Underflow

Severity: Low Risk

Context: CronV1Pool.sol#L1937

Description: Nothing prevents `lastVirtualOrderBlock` from advancing beyond the expiry of any given long-term swap, so the unchecked subtraction here is unsafe and *can* underflow. Since the resulting refund value will be extremely large due to the limited number of blocks that can elapse and the typical prices and decimals of tokens, the practical consequence will be a revert due to exceeding the pool and order balances. However, this could be used to steal funds if the value could be maliciously tuned, for example via another hypothetical bug that allowed the last virtual order block or the sales rate of an order to be manipulated to an arbitrary value.

Recommendation: Prevent the underflow in some way, either by explicitly checking for and reverting on attempts to cancel expired (i.e. filled) orders, or adding logic to skip refund calculation when expired orders are canceled.

Twamm: Fixed in commit `8e52bfa`.

Spearbit: Verified.

5.2.4 Function transferJoinEvent Permits Transfer-to-Self

Severity: Low Risk

Context: JoinEventLib.sol#L71

Description: Though the error code indicates the opposite intent, this check will permit transfer-to-self (`||` used instead of `&&`).

Recommendation: Use `&&` instead of `||`.

Twamm: Obsolete (join event logic removed).

Spearbit: Verified.

5.2.5 One-step owner change for factory owner

Severity: Low Risk

Context: CronV1PoolFactory.sol#L77

Description: The factory owner can be changed with a single transaction. As the factory owner is critical to managing the pool fees and other settings an incorrect address being set as the owner may result in unintended behaviors.

Recommendation: Consider implementing a two-step pattern for ownership changes.

Twamm: Resolved in commit `26058f7`.

Spearbit: Verified. Note that a one-step change is still possible using the `_direct` bool. However, the issue is mitigated.

5.2.6 Factory owner may front run large orders in order to extract fees

Severity: Low Risk

Context: CronV1Pool.sol#L1002, CronV1Pool.sol#L1531

Description: The factory owner may be able to front-run large trades in order to extract more fees if compromised or becomes malicious in one way or another.

Similarly, pausing may also allow for skipping the execution of virtual orders before exiting.

Recommendation: It is recommended to enforce that the factory owner is a 2/3 multisig.

Twamm: Addressed in commit `987f539`. Created a 2/3 multisig safe that is the factory owner and default pool admin.

Spearbit: Verified.

5.2.7 Join Events must be explicitly transfered to recipient after transfering Balancer Pool Tokens in order to realize the full value of the tokens

Severity: Low Risk

Context: CronV1Pool.sol#L1116

Description: Any user receiving LP tokens transferred to them must be explicitly transferred with a join event in order to redeem the full value of the LP tokens on exit, otherwise the address transferred to will automatically get the holding penalty when they try to exit the pool. Unless a protocol specifically implements `transferJoinEvent` function compatibility all LP tokens going through that protocol will be worth a fraction of their true value even after the holding period has elapsed.

Recommendation: Consider using OZ transfer hooks or overriding the balancer pool token's transfer function to automatically transfer joinEvents on token transfer. Although, this may increase the gas cost of an LP token transfer.

Twamm: Fixed. Join Event / Holding Period & Penalty completely removed in commit `3130e41`.

Spearbit: Verified.

5.2.8 Order Block Intervals(OBI) and Max Intervals are calculated with 14 second instead of 12 second block times

Severity: Low Risk

Context: Constants.sol#L100-L107, CronV1Pool.sol#L1783-L1789, CronV1Pool.sol#L100, VirtualOrders.sol#L382

Description: The CronV1Pool contract calculates both the Order Block Intervals (OBI) and the Max Intervals of the Stable/Liquid/Volatile pairs with 14 second block times. However, after the merge, 12 second block time is enforced by the Beacon Chain.

Recommendation: It is recommended to calculate the Order Block Intervals(OBI) and the Max Intervals with 12 second block times.

Twamm: Addressed in commit `74a4796` and commit `ac2280a5`.

Spearbit: Verified.

5.2.9 One-step status change for pool Admins

Severity: Low Risk

Context: CronV1Pool.sol#L864

Description: Admin status can be changed in a single transaction. This may result in unintended behaviour if the incorrect address is passed.

Recommendation: Consider implementing a two-step pattern.

Twamm: Acknowledged. We can always reset the admin in case the wrong address gets set, so 2 step process might be overkill and will increase contract size.

Spearbit: Acknowledged.

5.2.10 Incomplete token simulation in `CronV1Pool` due to missing `queryJoin` and `queryExit` functions

Severity: Low Risk

Context: CronV1Pool.sol#L76

Description: he CronV1Pool contract is missing the queryJoin and queryExit functions, which are significant for calculating `maxAmountsIn` and/or `minBptOut` on pool `joins`, and `minAmountsOut` and/or `maxBptIn` on pool `exits`, respectively. The ability to calculate these values is very important in order to ensure proper enforcement of slippage tolerances and mitigate the risk of sandwich attacks.

Recommendation: It is recommended to consider adapting the queryJoin and queryExit functions in the CronV1Pool contract.

Twamm: Sandwich protection for join (and testing for exit) was added as a mitigation for the *"Overpayment of one side of LP Pair onJoinPool due to sandwich or user error"* finding. The team will add `queryJoin/queryExit` in peripheries contracts.

Spearbit: Verified.

5.2.11 A partner can trigger ROC update

Severity: Low Risk

Context: CronV1Pool.sol#L1073

Description: A partner can trigger rook update if they return rook's current list within an update.

- Scenario

A partner calls `updateArbitrageList`, the `IArbitrageurList(currentList).nextList()` returns rook's `rookPartnerContractAddr` and gets updated, the partner calls `updateArbitrageList` again, so this time `isRook` will be true.

Recommendation: Consider adding a check if the partner returns `rookPartnerContractAddr` to revert.

Twamm: Addressed in commit `850eb42` by removing the Rook specific implementation.

Spearbit: Verified.

5.2.12 Approved relayer can steal cron's fees

Severity: Low Risk

Context: CronV1Pool.sol#L1826

Description: A relayer within Balancer is set per vault per address.

If `feeAddr` will ever add a relayer within the balancer vault, the relayer can call `exitPool` with a recipient of their choice, and the check on line 225 will pass as the sender will still be `feeAddr` but the true `msg.sender` is the relayer.

Recommendation: Consider setting as `feeAddr` a multisig between trusted parties to limit the possibility of approving a relayer within the Balancer Vault for the `feeAddr`. There is no immediate action that can be done as the approved relayer functionality is part of the Balancer functionality.

Twamm: The `feeAddr` will be an address controlled by multisig. No additional logic will be introduced in the smart contract to prohibit relayers as this will force `feeAddr` wallet to be an EOA and limit usability.

Spearbit: Acknowledged.

5.2.13 Price Path Due To Long-Term Orders Neglected In Oracle Updates

Severity: Low Risk

Context: CronV1Pool.sol#L1564

Description: The `_updateOracle()` function takes its price sample as the final price after virtual order execution for whatever time period has elapsed since the last `join/exit/swap`. Since the price changes continuously during that interval if there are long-term orders active (unlike in Uniswap v2 where the price is constant between swaps), this is inaccurate - strictly speaking, one should integrate over the price curve as defined by LT orders to get a correct sample. The longer the interval, the greater the potential for inaccuracy.

Recommendation: Consider accumulating oracle updates within the virtual order execution loop, or at least clearly document the potential for tracking errors in prices.

Twamm: Fixed in commit `b2339e6` , commit `2b48f50`, and commit `08f96a1`.

Spearbit: Verified.

5.2.14 Vulnerabilities noted from `npm audit`

Severity: Low Risk

Context: package.json#L37-L81

Description: `npm audit` notes: `76 vulnerabilities (5 low, 16 moderate, 27 high, 28 critical).`

Recommendation: Remove any unused dependencies and update or patch them elsewhere.

Twamm: Resolved prod issues `npm audit fix --only=prod` in commit `8bd2644`.

Spearbit: Fixed.

5.3 Gas Optimization

5.3.1 Optimization: Merge CronV1Pool.sol & VirtualOrders.sol (Changes from dev team added to audit.)

Severity: Gas Optimization

Context: CronV1Pool.sol#L29

Description: A lot of needless parameter passing is done to accommodate the file barrier between CronV1Pool & VirtualOrdersLib, which is an internal library. Some parameters are actually immutable variables.

Recommendation: Merging these two files and removing extra members from ExecuteVirtualOrdersMem struct will reduce gas use and contract size.

Twamm: Addressed in commit `0d14f0a`.

Spearbit: Verified.

5.3.2 Receive sorted tokens at creation to reduce complexity

Severity: Gas Optimization

Context: CronV1Pool.sol#L491

Description: Currently, when a pool is created, within the constructor, logic is implemented to determine if the tokens are sorted by address. A requirement that is needed for Balancer Pool creation. This logic adds unnecessary gas consumption and complexity throughout the contract as every time amounts are retrieved from balancer, the Cron Pool must check the order of the tokens and make sure that the difference between sorted (Balancer) and unsorted (Cron) token addresses is handled.

An example can be seen in onJoinPool

```
uint256 token0InU112 = amountsInU112[TOKEN0_IDX];
uint256 token1InU112 = amountsInU112[TOKEN1_IDX];
```

Where the `amountsInU112` are retrieved from the balancer as a sorted array, index 0 == token0 and index 1 == token1, but on the Cron side, we must make sure that we retrieved the correct amount based on the tokens being sent as sorted or not when the pool was created.

Recommendation: Make an explicit requirement within the constructor, in order for the tokens to be sorted

```
require("_token0Inst < _token1Inst", ....);
```

Twamm: Resolved in commit `26058f77`.

Spearbit: Verified.

5.3.3 Remove double reentrancy checks

Severity: Gas Optimization

Context: CronV1Pool.onSwap, CronV1Pool.onJoinPool, CronV1Pool.onExitPool

Description: A number of `CronV1Pool` functions include reentrancy checks, however, they are only callable from a Balancer Vault function that already has a reentrancy check.

Recommendation: Exercise caution as changes to either Balancer or Cron may re-introduce risks.

In cases where the only entry point to a `CronV1Pool` function is a Balancer Vault and the vault's function includes a `nonReentrant` modifier, the extra `nonReentrant` modifier on the `CronV1Pool` function may be removed.

Twamm: Addressed in commit `756576a` and commit `731059b`.

Spearbit: Verified.

5.3.4 TWAMM Formula Computation Can Be Made Correct-By-Construction and Optimized

Severity: Gas Optimization

Context: VirtualOrders.sol#L331-L336

Description: The linked lines are the core calculation of TWAMM virtual order execution. They involve checked arithmetic in the form of underflow-checked subtractions; there is thus a theoretical risk that rounding error could lead to a "freezing" of a TWAMM pool. One of the subtractions, that for `token1OutU112`, is already "correct-by-construction", i.e. it can never underflow. The calculation of `token0OutU112` can be reformulated to be explicitly safe as well; the following overall refactoring is suggested:

```
uint256 ammEndToken0 = (token1ReserveU112 * sum0) / sum1;
uint256 ammEndToken1 = (token0ReserveU112 * sum1) / sum0;
token0ReserveU112 = ammEndToken0;
token1ReserveU112 = ammEndToken1;

token0OutU112 = sum0 - ammEndToken0;
token1OutU112 = sum1 - ammEndToken1;
```

Both output calculations are now of the form $x - (x * y)/(y + z)$ for non-negative x, y, and z, allowing subtraction operations to be unchecked, which is both a gas optimization and gives confidence the calculation cannot freeze up unexpectedly due to an underflow. Replacement of `divDown` by `/` gives equivalent semantics with lower overhead. An additional advantage of this formulation is its manifest symmetry under $0 < - > 1$ interchange; this serves as a useful heuristic check on the computation, as it should possess the same symmetry as the invariant.

Recommendation: Reformulate as suggested above or in an analogous fashion.

Twamm: Fixed in commit `eb957842`.

Spearbit: Verified.

5.3.5 Gas Optimizations In Bit Packing Functions

Severity: Gas Optimization

Context: BitPacking.sol

Description: The bit packing operations are heavily used throughout the gas-critical swap code path, the optimization of which was flagged as high-priority by the CronFi team. Thus they were carefully reviewed not just for correctness, but also for gas optimization.

L119: unnecessary `&` due to check on L116 L175: could hardcode `clearMask` L203: could hardcode `clearMask` L240: could hardcode `clearMask` L241: unnecessary `&` due to check on line 237 L242: unnecessary `&` due to check on line 238 L292: could hardcode `clearMask` L328: could hardcode `clearMask` L343: unnecessary to mask when `_isWord0 == true` L359: unnecessary `&` operations due to checks on lines 356 and 357 L372: unnecessary masking L389: could hardcode `clearMask` L390: unnecessary `&` due to check on L387 L415: could hardcode `clearMask` L416: unnecessary `&` operation due to check on line 413 L437: unnecessary `clearMask` L438: unnecessary `&` due to check on line 435 L464: could hardcode `clearMask` L465: unnecessary `&` due to check on line 462

Additionally, the following code pattern appears in multiple places:

```
requireErrCode(increment <= CONST, CronErrors.INCREMENT_TOO_LARGE);
value += increment;
requireErrCode(value <= CONST, CronErrors.OVERFLOW);
```

Unless there's a particular reason to want to detect a too-large increment separately from an overflow, these patterns could all be simplified to

```
requireErrCode(CONST - value >= increment, CronErrors.OVERFLOW);
value += increment;
```

as any increment greater than `CONST` will cause overflow anyway and `value` is always in the correct range by construction. This allows `CronErrors.INCREMENT_TOO_LARGE` to be removed as well.

Recommendation: If desired, these small optimizations can be made safely.

Twamm: Fixed in commit cf80681. Error code optimization implemented as suggested, except for the `incrementPairWithClampU96` method, absolute overflow of the container (U256) is checked instead of U96 because the clamping action would be subverted if the check were against MAX_U96. Measured contract size reduced by 187 bytes.

Spearbit: Verified.

5.3.6 Using extra storage slot to store two mappings of the same information

Severity: Gas Optimization

Context: CronV1PoolFactory.sol#L69

Description: A second storage slot is used to store a duplicate mapping of the same token pair but in reverse order. If the tokens are sorted in a getter function then the second mapping does not need to be used.

Recommendation: Remove the use of the second mapping and create a getter function that sorts the tokens before returning the mapping.

Alternatively, if `CREATE2` was used instead it could be used to calculate the deployment of a pool's address.

Twamm: Resolved in: commit `26058f7`.

Spearbit: Fixed.

5.3.7 Gas optimizations within `_executeVirtualOrders` function

Severity: Gas Optimization

Context: CronV1Pool.sol#L1519:L1569

Description: Within the `_executeVirtualOrders` function there are a few gas optimizations that can be applied to reduce the contract size and gas consumed while the function is called.

`(!(virtualOrders.lastVirtualOrderBlock < _maxBlock && _maxBlock < block.number))` Is equivalent with: `(virtualOrders.lastVirtualOrderBlock >= _maxBlock || _maxBlock >= block.number)`

This means that this always enters if `_maxBlock == block.number` which will result in unnecessary gas consumption.

If cron fee is enabled, `evoMem.feeShiftU3` will have a value meaning that the check on line 1536 is obsolete. Removing that check and the retrieve from storage will save gas.

Recommendation: Consider implementing the above gas optimizations.

Twamm: Resolved in commit `3ae7fce` and commit `6826034`.

Spearbit: Verified.

5.3.8 Initializing with default value is consuming unnecessary gas

Severity: Gas Optimization

Context: CronV1Pool.sol#L2102

Description: Every variable declaration followed by initialization with a default value is gas consuming and obsolete. The provided line within the context is just an example.

Recommendation: Consider going through the contracts and deleting any default initialization to save gas.

Twamm: Addressed.

Spearbit: Verified.

5.3.9 Factory requirement can be circumvented within the constructor

Severity: Gas Optimization

Context: CronV1Pool.sol#L484

Description: The constructor checks if the `_factory` parameter is the `msg.sender`. This behavior was, at first, created so that only the factory would be able to deploy pools. The check on line 484 is obsolete as pools deployed via the factory, will always have `msg.sender == factory` address, making the `_factory` parameter obsolete as well.

Recommendation: Consider if it is necessary to include this check and just set the FACTORY as `msg.sender` to save gas.

Twamm: Addressed in commit `26058f7`.

Spearbit: Verified.

5.4 Informational

5.4.1 Usability: added `remove`, `set` pool functionality to CronV1PoolFactory (Changes from dev team added to audit.)

Severity: Informational

Context: CronV1PoolFactory.sol#L49

Description: Conversations with the audit team indicated functions were needed to manage pool mappings post-creation in the event that a pool needed to be deprecated or replaced.

Recommendation: Add a set and remove function to accomplish this.

Twamm: Addressed in commit `bad2211`.

Spearbit: Verified.

5.4.2 Virtual oracle getter--gets oracle value at block > lvob (Changes from dev team added to audit.)

Severity: Informational

Context: CronV1Pool.sol#L274

Description: Through the audit process, sufficient contract space became available to add an oracle getter convenience that returns the oracle values and timestamps. However, this leaves the problem of not being able to get the oracle price at the current block in a pool with low volume but virtual orders active.

Recommendation: Introduce a way to get the virtual value of the oracle at a specified block number.

Twamm: Addressed in commit `2b48f50`.

Spearbit: Verified.

5.4.3 Loss of assets due to rounding during `_longTermSwap`

Severity: Informational

Context: CronV1Pool.sol#L1752

Description: When a long term swap (LT) is created, the selling rate for that LT is set based on the amount and the number of blocks that order will be traded for.

```
uint256 lastExpiryBlock = block.number - (block.number % ORDER_BLOCK_INTERVAL);
uint256 orderExpiry = ORDER_BLOCK_INTERVAL * (_orderIntervals + 1) + lastExpiryBlock; // +1 protects
↪   from div 0
uint256 tradeBlocks = orderExpiry - block.number;
uint256 sellingRateU112 = _amountInU112 / tradeBlocks;
```

During the computation of the number of blocks, the order must trade for, defined as `tradeBlocks`, the order expiry is computed from the last expiry block based on the OBI (Order Block Interval).

If tradeBlocks is big enough (it can be a max of 176102 based on the `STABLE_MAX_INTERVALS`), then `sellingRateU112` will suffer a loss due to solidity rounding down behavior. This is a manageable loss for tokens with big decimals but for tokens with low decimals, will create quite an impact.

E.g. wrapped BTC has 8 decimals. the MAX_ORDER_INTERVALS can be max 176102 as per stable max intervals defined within the constants. that being said a user can lose quite a significant value of BTC: 0.00176101

This issue is marked as Informational severity as the amount lost might not be that significant. This can change in the future if the token being LTed has a big value and a small number of decimals.

Recommendation: Consider making this as transparent as possible to the users via frontend, additional documentation might help as well when 3rd parties will want to integrate with the Cron Pool.

Twamm: Acknowledged.

Spearbit: Acknowledged.

5.4.4 Inaccuracies in Comments

Severity: Informational

Context: [1] BitPacking.sol#L52 [2] BitPacking.sol#L325 [3] JoinEventLib.sol#L16 [4] CronV1Pool.sol#L456 [5] Structs.sol#L193-L194 [6] VirtualOrders.sol#L64-L66 [7] VirtualOrders.sol#L123-124 [8] CronV1Pool.sol#L349

Description: A number of minor inaccuracies were discovered in comments that could impact the comprehensibility of the code to future maintainers, integrators, and extenders.

[1] `bit-0` should be `bit-1` [2] `less than` should be `at most` [3] `Maximally Extracted Value` should be `Maximal Extractable Value` see flashbots.net [4] `Maximally Extracted Value` should be `Maximal Extractable Value` see flashbots.net [5] on these lines `unsold` should be `sold` [6] These comments are not applicable to the code block below them, as they mention looping but no looping is done; it seems they were copied over from the loop

on line 54. [7] These comments are not applicable to the code block below them, as they mention looping but no looping is done; it seems they were copied over from the loop on line 111. [8] `omitted` should be `emitted`

Recommendation: Correct or otherwise fix the indicated issues in the comments.

Twamm: [1] addressed in commit `cf80681` [2] addressed in commit `cf80681` [3] addressed in commit `0e2cc05`; also, join events removed [4] addressed in commit `ec2d92b` [5] addressed in commit `c903d4d` [6] addressed in commit `d632f85` [7] addressed in commit `d632f85` [8] addressed in commit `b1e11d6`

Spearbit: Verified.

5.4.5 Unsupported `SwapKind.GIVEN_OUT` may limit the compatibility with Balancer

Severity: Informational

Context: CronV1Pool.sol#L599

Description: The Balancer protocol utilizes two types of swaps for its functionality - `GIVEN_IN` and `GIVEN_OUT`.

- `GIVEN_IN` specifies the minimum amount of tokens a user would accept to receive from the swap.
- `GIVEN_OUT` specifies the maximum amount of tokens a user would accept to send for the swap.

However, the `onSwap` function of the CronV1Pool contract only accepts the `IVault.SwapKind.GIVEN_IN` value as the `IVault.SwapKind` field of the `SwapRequest` struct.

The unsupported `SwapKind.GIVEN_OUT` may limit the compatibility with Balancer on the Batch Swaps and the Smart Order Router functionality, as a single `SwapKind` is given as an argument.

Recommendation: In order to increase compatibility with Balancer, it is recommended to include support for the `SwapKind.GIVEN_OUT` functionality in the `onSwap` function.

Twamm: Acknowledged.

Spearbit: Acknowledged.

5.4.6 A pool's first LP will always take a minor loss on the value of their liquidity

Severity: Informational

Context: CronV1Pool.sol#L2046

Description: The first liquidity provider for a pool will always take a small loss on the value of their tokens deposited into the pool because 1000 balancer pool tokens are minted to the zero address on the initial deposit. As most tokens have 18 decimal places, this value would be negligible in most cases, however, for tokens with a high value and small decimals the effects may be more apparent.

Recommendation: Document and ensure this behavior is known.

Another option is to make the minimum liquidity configurable.

Twamm: A warning has been written in the contract's comments. Addressed in commit `839ee51`.

Spearbit: Verified.

5.4.7 The `_withdrawCronFees` functions should revert if no fees to withdraw

Severity: Informational

Context: CronV1Pool.sol#L1840

Description: The `_withdrawCronFees` checks if there are any Cron Fees that need to be withdrawn, currently, this function does not revert in case there are no fees.

Recommendation: We recommend reverting in case there are no fees to withdraw, this will prevent unnecessary gas consumption if an admin calls this function.

Twamm: Addressed in commit `c4dfaa9`.

Spearbit: Verified.

5.4.8 Extend the functionality of `getVirtualReserves` function

Severity: Informational

Context: CronV1Pool.sol#L1250

Description: The `getVirtualReserves` function is a critical function within the Cron Pool. We recommend expanding a bit this function in order to make it more useful for 3rd parties to integrate with the pool. One suggestion is that the function is not parametrized with block number, in case something happens and this function runs out of gas because of the block.number - LVOB is too big, a block number parameter can save on debugging at least.

A second suggestion is to add an extra parameter for isPaused requirement, as even if the pool is paused, a user might want to see what the virtual balance would be if the orders would have been executed.

A third suggestion is considering creating a helper contract to enable more options of the reservers/sale rates, e.g. getting the sale rate at a specific expiry block.

Recommendation: Consider if the above suggestions are valid for better integration of the 3rd parties with the Cron Finance Pools.

Twamm: Addressed in commit `d2cccce`.

Spearbit: Verified.

5.4.9 Loss of tokens if swapping to pool with 0 liquidity and 0 slippage

Severity: Informational

Context: CronV1Pool.sol#L596

Description: Although extremely unlikely it is possible for a user to swap tokens into the pool before the pool has any liquidity, if they do not pass a slippage limit on the balancer swap function, those tokens will be lost.

Recommendation: In the UI do not allow users to swap into a pool if there is no liquidity present.

Twamm: Acknowledged.

Spearbit: Acknowledged.

5.4.10 The `ArbPartnerStatusChange` event is not used

Severity: Informational

Context: CronV1Pool.sol#L359

Description: The `ArbPartnerStatusChange` event is not used within the codebase.

Recommendation: Remove the unused event.

Twamm: Addressed in commit `b1fcea7`.

Spearbit: Verified.

5.4.11 Simplify RookSwap & PartnerSwap mechanisms

Severity: Informational

Context: CronV1Pool.sol#L619

Description: RookSwap is a subset of the generic partner swap and introduces additional contract size, attack surface area, and complexity.

Recommendation: Remove RookSwap. Have partners use the PartnerSwap mechanism.

Twamm: Addressed in commit `850eb42`.

Spearbit: Verified.

5.4.12 The `getOrderIds` view is limited on retrieving a small number of orders

Severity: Informational

Context: CronV1Pool.sol#L1153

Description: The `getOrderIds` is a function used to get for a specific `owner`, the order data. The function is paginated and the max number of results per page is limited to 100, by the `MAX_RESULTS`. This can introduce a bad UX for owners with a lot of orders (like 3rd parties that would want to integrate with the protocol) as they have to call this function multiple times to get the total amount of orders.

Recommendation: This being a view function, not consuming on-chain gas, consider removing the `MAX_RESULTS` constraint and just introduce a parameter, in order to give the flexibility to the users to retrieve as many orders as they want.

Twamm: Addressed in commit `76fc327` and in commit `106243d`. Amended/Squashed from these two due to omitting changes to foundry/interfaces:

Spearbit: Verified.

5.4.13 `WithdrawLongTermSwap` event is ambiguous--cancel or withdraw? (Changes from dev team added to audit.)

Severity: Informational

Context: CronV1Pool.sol#L324

Description: The `WithdrawLongTermSwap` event is overloaded--it is emitted for both Cancel Long-Term (LT) swap transactions and Withdraw LT swap transactions. This was done to reduce contract size.

Unfortunately, in a corner case, it may not be possible to discern if a user was performing a Cancel or a Withdraw from the emitted event log. This will happen when a user performs a Cancel LT swap transaction right at the end of their order on the `expiryBlock`--at that point, there may be no refund available, only the proceeds of the transaction.

This code from `CronV1Pool.sol::_withdrawLongTermSwap` illustrates the situation; consider `expiryBlock == lastVirtualOrderBlock`:

```
if (_cancel) {
  // Calculate unsold amount to refund:
  //
  // NOTE: Must use last virtual order block, not block.number to yield correct result
  //       when pool is paused.
  ...
  refundU112 = (expiryBlock - virtualOrders.lastVirtualOrderBlock) * salesRateU112;
}
```

Recommendation: Create a test case that confirms this. If this is indeed an issue--or even if it is not, consider modifying the emitted event to output the NULL address as the refund token when a Withdraw LT swap transaction is performed.

Twamm: Addressed in commit `980a598`.

Spearbit: Verified.

5.4.14 Expose Last Virtual Order Block (LVOB) externally. (Changes from dev team added to audit.)

Severity: Informational

Context: CronV1Pool.sol#L1263

Description: Changes from the development team added to the audit.

The contract does not expose the Last Virtual Order Block (LVOB) to users. In discussion with customers/partners, an interest in estimating gas for calls and understanding the activity, etc. of the pool was indicated. This would be challenging without the LVOB, requiring inefficient solutions like monitoring the log stream for transactions that update the LVOB or decoding chain state. Furthermore, this would also be very challenging for contracts wishing to access this information.

Recommendation: Create an external view getter: `getLastVirtualOrderBlock`. It will return a uint256 containing the value of `struct VirtualOrders.lastVirtualOrderBlock`.

Twamm: Addressed in commit `4fea4581`.

Spearbit: Verified.

5.4.15 Remove holding period and penalty. (Changes from dev team added to audit.)

Severity: Informational

Context: Entire Product

Description: The holding period and penalty feature introduced considerable attack surface area, gas use, and UX complexity to prevent a problem that is difficult to accurately quantify and can be addressed in other ways (chiefly by increasing the frequency of MEV reward distribution and the randomness of the distribution).

Recommendation: Remove the holding period and penalty feature.

Twamm: Join Event / Holding Period & Penalty completely removed in commit `3130e41`.

Spearbit: Verified.

6 Appendix

The following issues have been raised by the TWAMM team post-engagement and have yet to be scrutinized by the Spearbit team. They are presently being included within this section for the purpose of ensuring visibility.

6.1 Update April 11th of 2023

The TWAMM team has extended the review period starting April 5th and running until April 7th and as a result all issues from the previous review have been verified.

6.1.1 Usability: getVirtualReserves returns values in different contexts via a proprietary internal struct (Changes from dev team added to audit.)

Context: CronV1Pool.sol#L1250

Description: The `getVirtualReserves` method returns a `blockNumber` and the `ExecuteVirtualOrderMem` struct at present. The struct contains the virtual reserves and then a bunch of virtual differences for pool accounting like orders, proceeds and fees. This is confusing to external users and can be cleaned up to be more straightforward and usable.

Recommendation: Return all values as virtual values, not some as virtual values and some as virtual differences.

Twamm: Addressed in commit `f456746`.

Spearbit:

- The comments on `_getVirtualReserves` state

```
///                IMPORTANT Note above that struct values for token0ReserveU112 and token1ReserveU112
///                          are the virtual values of the reserves at the current block, whereas other
///                          values in the ExecVirtualOrdersMem struct are the DIFFERENCES from the
///                          state of their respective values and their virtual values.
```

However, this is inaccurate--instead of current block, the return value blockNumber should be specified as the block to which values have been computed.

- The comments on `getVirtualReserves` state

```
/// @return token0ProceedsU112 virtual amount of Token0 purchased from the pool in from LT swap orders
///                            at blockNumber.
/// @return token1ProceedsU112 virtual amount of Token1 purchased from the pool in from LT swap orders
///                            at blockNumber.
```

Consider replacing in from LT swap orders with a less confusing phrasing like by LT swap orders.

Comments on comments aside, code change LGTM.

Twamm: Addressed in commit bf350da.

Spearbit: Verified.

6.1.2 LT Swap DAO Realities Necessitate Delegate Role (Changes from dev team added to audit.)

Context: CronV1Pool.sol#L1721

Description: Getting the attention of a DAO (i.e. an actionable vote) to adjust an ongoing LT position in the face of changing market conditions may prove challenging for some DAO customers.

Recommendation: In conversations with partners, this was brought up multiple times and it was requested that DAOs could specify a delegate when issuing an LT Swap. The delegate would be able to:

- Withdraw or cancel the LT Swap as long as the "recipient" address that is the destination of the funds is set to the original owner.

Twamm: Update to fix null addr vuln commit e96a0de.

Spearbit: Fix commit LGTM. Small consideration: it is unclear whether it's ever valid for _sender to be the zero address, but if it isn't, the zero address check could be done in all cases. This doesn't seem to be necessary, but it could be a good precaution if it does not interfere with valid use cases.

Twamm: Agreed and addressed in commit 76f948a

Spearbit: Verified.

6.1.3 executeVirtualOrdersView does not handle final interval update correctly when new lvob == block.number

Context: VirtualOrders.sol#L126

Description: Execute Virtual Orders (EVO) View runs updates in a loop and then runs a final update if the Last Virtual Order Block (LVOB) is not equal to block.number. Presently it's always running the final update because it's comparing to LVOB in state, not the value updated in memory.

Recommendation: Likely low impact as this issue resulted in the code always running, however the comparison needs to be made to the new last virtual order block, not the one in state (which cannot be permuted by the view function). Replace with comparison to updated last virtual order block in `ExecuteVirtualOrdersMem`.

Twamm: Addressed in commit 2b48f50.

Spearbit: Looks like the fix was actually in this commit commit b2339e6.

Out of curiosity I traced what the effect of this would be, and looks like the only cost is extra computation--in the case where the code is run erroneously, `intervalBlocks` will be zero in `_executeVirtualTradesAndOrderExpiries()`, resulting in all token inputs being zero, which further implies no changes to reserves, fees, or scaled proceeds.

Verified.

Additionally, during the 2 day extension period the following contracts were added to the scope:

1. `contracts/twault/periphery/CronV1Relayer.sol`
2. `contracts/twault/interfaces/ICronV1Relayer.sol`
3. `contracts/twault/periphery/CronV1Actions.sol`
4. `contracts/twault/interfaces/IBaseRelayerLibrary.sol`

6.1.4 Relayer Can Be Sent ETH to DOS Incautious Integrations

Severity: Low Risk

Context: CronV1Relayer.sol#L77

Description: The CronV1Relayer contract checks that any sender of ETH (in contexts outside of public `payable` function calls) is the Balancer Vault, intending to limit ETH receipt to valid CronFi pool interactions. However, ETH sent using the SELFDESTRUCT opcode will not trigger this check. The check can also be passed in a manner that arguably violates intent if someone specifies the relayer's address as the recipient of funds from an unrelated (to CronFi pools) Balancer Vault interaction. Thus the contract can be maliciously given a non-zero ETH balance. Since the relayer sends any free ETH in its account to the caller at the end of public function calls, smart contract integrations may assume the relayer will never send them ETH and thus revert-on-receipt-of-ETH (the default behavior for Solidity contracts). This would allow an attacker to perform denial-of-service on such integrations by repeatedly sending 10^{-18} ETH via SELFDESTRUCT or other Balancer pools to the relayer as needed.

Recommendation: One option is to clearly document that the CronV1Relayer cannot be guaranteed to not send ETH back to a caller, even if they are sending no ETH themselves. Or, since CronFi TWAMM pools never use raw ETH as a token, support for refunding raw ETH could simply be removed as there is no expected scenario where the relayer should need to receive raw ETH.

Twamm: ETH refunding behavior removed in commit aec9109.

Spearbit: Verified.

6.1.5 Unreachable Code in CronV1Relayer

Severity: Gas Optimization

Context: CronV1Relayer.sol#L99-L101

Description: The requirement that `msg.sender == address(this)` on line 89 implies that the body of this `if` statement will never be executed. Consider removing it, leaving a comment for documentation if desired.

Recommendation: Remove unreachable code, leaving documentation if desired.

Twamm: Code disabled in commit 2afb480 and issue made obsolete by commit 6d9be82.

Spearbit: Verified.

6.1.6 Considerations Around multicall Pattern

Severity: Gas Optimization

Context: CronV1Relayer.sol#L89

Description: The pattern used by the relayer to conform to Balancer's documentation is somewhat unnatural and inefficient. This involves limiting the caller of the `multicall` function to be only the relayer contract itself, and having the universally accessible functions of the relayer do an extra call into `multicall`. A number of discussions between the CronFi, Spearbit, and Balancer teams suggested that this was unnecessary as the CronFi relayer does not support batched operations.

Recommendation: Simplify the relayer implementation as much as possible while retaining conformance to Balancer requirements.

Twamm: Call-to-self `multicall` pattern removed in commit 12e0f6f and commit 6d9be829.

Spearbit: Verified.

6.1.7 Unnecessary Non-negativity Checks

Severity: Gas Optimization

Context: [1] CronV1Actions.sol#L497 [2] CronV1Relayer.sol#L207

Description: On these two lines, `_poolType` is an unsigned integer, so the checks for `_poolType >= 0` are vacuous and can be removed.

Recommendation: Remove all unnecessary non-negativity checks.

Twamm: Addressed in commit 3f2ad8515 and commit dc19c5105.

Spearbit: Verified.

6.1.8 Unnecessary ERC-20 Approval

Severity: Gas Optimization

Context: CronV1Actions.sol#L275

Description: This ERC-20 approval is unnecessary because the Balancer Vault will pull the tokens directly from the original sender.

Recommendation: Remove the unnecessary approval.

Twamm: Addressed in commit 04bd832fc.

Spearbit: Verified.

6.1.9 Unnecessary Checks in multicall

Severity: Gas Optimization

Context: CronV1Relayer.sol#L91-92

Description: Since `multicall` is only callable by a limited set of functions of the relayer itself, which all construct arguments correctly, these checks--while they may have been useful during development--are not necessary.

Recommendation: Remove the unnecessary checks.

Twamm: Obsolete `multicall` pattern removed in commit 12e0f6f95 and commit 6d9be8294.

Spearbit: Verified.

6.1.10 Debugging Import Can Be Removed

Severity: Informational

Context: CronV1Relayer.sol#L8

Description: This development framework import was presumably for debugging and should be removed.

Recommendation: Remove this import.

Twamm: Addressed in commit 3261991.

Spearbit: Verified.

6.1.11 Redundant Checks

Severity: Informational

Context: CronV1Actions.sol#L519

Description: The periphery contracts do many checks that are redundant with checks that will be enforced either by the Balancer Vault or the CronFi pool implementation (only one example is given here for brevity). These will provide more granular debugging information, and will generally short-circuit failing transactions faster (saving callers gas on failures); however, they add unnecessary gas costs to successful transactions and increase code complexity without any safety benefit.

Recommendation: Consider whether the various redundant checks are worthwhile given the trade-offs mentioned above.

Spearbit: Acknowledged.

6.1.12 Indirect Calls to Reentrancy Checks Are a Risky Pattern

Severity: Informational

Context: CronV1Pool.sol#L2256

Description: Doing the Balancer Vault reentrancy check in a sub-function like this makes it difficult to 1) see that the check is obviously happening where it needs to, and 2) increases the risk that future refactoring could remove it from functions where it is needed.

Recommendation: Consider putting the reentrancy check explicitly in affected functions, or document that it is done in a sub-function.

Twamm: Documented in commit 1747466.

Spearbit: Verified.