Sommelier A-4

Security Audit

January 11, 2023
Version 1.0.0

Presented by 0xMacro

Table of Contents

Introduction

This document includes the results of the security audit for Sommelier Finance's smart contract code as found in the section titled ‘Source Code’. The security audit was performed by the Macro security team from November 28, 2022 to December 23, 2022.

The purpose of this audit is to review the source code of certain Sommelier Finance Solidity contracts, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

Disclaimer: While Macro’s review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.

Overall Assessment

The following is an aggregation of issues found by the Macro Audit team:

Severity	Count	Acknowledged	Won't Do	Addressed
High	5	-	-	5
Medium	8	3	-	5
Low	2	-	-	2
Code Quality	7	2	-	5

Sommelier Finance was quick to respond to these issues.

Specification

Our understanding of the specification was based on the following sources:

- Discussions on Telegram with the Sommelier Finance team.
- Publicly available Sommelier Finance documentation.

Source Code

The following source code was reviewed during the audit:

- **Repository:** cellar-contracts
- **Commit Hash:** `760dc9cff074b379f1fd18b3d5c646761050ab34`

Specifically, we audited the following contracts within this repository:

Contract	SHA256
src/CellarFactory.sol	a29bca2fae4dc622b90346b1f7852453e3503a840701be2a7464e27b2895d385
src/CellarRouter.sol	13e6b5d9e57feb9b67551c78212c7b89888a26de8ffe66b98c0dffd603394dad
src/Registry.sol	1f2b7370db86782a53f80550f488d895d9e003f03fd16ccbac49d78ec688aa48
src/base/Cellar.sol	7216b4241f3d7498590837d192defc08cc772e04017850fd97a4cf6a122a15c6
src/base/CellarInitializable.sol	c08eadbc13ed403dc0bda19bbfb2c2ea19dda7b7fbba0e49b4b014a691556be4
src/base/ERC20.sol	034ad547f0d49172d9675eeda9e7455d848a93e3488f651a58918e7abd0b31c0
src/base/ERC4626.sol	7339b94d56e93300c814ab96cc984f3f3411d11160a5937b6ec11c02962534bd
src/base/Multicall.sol	cbe4ba9c787b262b1b77a83a9969ff9a716d88f09048d175c1a0d00ee4414e14
src/base/SafeTransferLib.sol	83a15e116064dc682a367cc52c5d45d8e1d951534bf4bc55c966dfd626f46ea1

Contract	SHA256
src/interfaces/ICellarRouter.sol	af2afbee5468fabae1780eb7e576714edcc66bb10e0ae6bb650ca644a309bbe5
src/interfaces/ICellarRouterV1_5.sol	806b8683bafde540975057eed16fa387b6948bd39c4a4920438c16d90034e72d
src/interfaces/ICellarStaking.sol	66750c8a92b3c8a68fef6e3ac1c251803a3b8450a718f46f7621496751b8d967
src/interfaces/ICellarV1_5.sol	0dc13ae1f9a72dfbf71d34a400b2bc245c5b91d4f4301f3b45ecddad5cd5c6ce
src/interfaces/IMulticall.sol	0d976a972ff63255c5b191b610d7aabe2c43e2918db8e37ee15a28ed3f958586
src/interfaces/external/DataTypes.sol	355a17582931dd2eb6fdac9474d2145f41c660cf547624f913e5693a724a1a72
src/interfaces/external/IAaveToken.sol	6d94860fa2048a78e972a60df6869f95c6c9816ca77ac1295221c3977451b028
src/interfaces/external/IChainlinkAggregator.sol	4a3a658744c35c28be6f4bd4984427e3383173a2f7186a89f3ca10dafbbb91df
src/interfaces/external/ICurveFi.sol	0ed58af3419766ab14c9c3554614e2b24ef41af0111cc69f4928f64088519092
src/interfaces/external/ICurvePool.sol	18cf0d11b734b6ad0b67244ee013ad78a9bc06d62fd49781a4717d37cb1e564c
src/interfaces/external/IGravity.sol	e09c292aa38c53f6282949761962ef1b8b22f9ea00201477b515a3b44dd37c90
src/interfaces/external/IPool.sol	964f63843ce11c53aaf1e015fb5f7042089687ff4deada92011c948a7b5cf8ff
src/interfaces/external/IPoolAddressesProvider.sol	7c84dacfbe52dcab2b6eab5d007568333a7377278a5b634204e1de7c99621bf3

Contract	SHA256
src/interfaces/external/IUniswapV2Router02.sol	b1b5ea5db6e598cf9396b64754c4fd2397ff86c6c870b8daf870c5ec427477f6
src/interfaces/external/IUniswapV3Router.sol	07e54f895d2e96a922fd26ed7936c3ca432047815f40b9651d613cb73fcb8d81
src/modules/adaptors/Aave/AaveATokenAdaptor.sol	ca370339d135937d8416110f324034c0306b7ea6be57e34843ea7a653dfdd8fa
src/modules/adaptors/Aave/AaveDebtTokenAdaptor.sol	26134deb7d44f89d35518cae52e99d03afa061f3f685e367323ab8cac0369562
src/modules/adaptors/BaseAdaptor.sol	e593629ffda30369d8fe1fcb36d997f0930ebb783ed616589c2974bbc9fc05d2
src/modules/adaptors/Compound/CTokenAdaptor.sol	a1811b8d3e9f6ee312564bf18bc1995791c592d54c50630c9eaef8d925ce3499
src/modules/adaptors/ERC20Adaptor.sol	b9fe6e24df7c28a984c0cb1a665d514d586659d0d084542b06bfd7d6ece8ec42
src/modules/adaptors/Sommelier/CellarAdaptor.sol	3298b08a490b75023836191f7d420abe00032a62b157ce89096fec509571b07a
src/modules/adaptors/UniSwap/UniswapV3Adaptor.sol	f99dab91dc3cd744485c88bf4dd5d9bd99c6225984ec8babe33b64ab5d7fc3b7
src/modules/adaptors/VestingSimpleAdaptor.sol	a8bf1e395d5c1460238ddef2f4885707da66a625d647d4eabc19932a46451773
src/modules/price-router/PriceRouter.sol	e34df27205ee255508264c0d4f36c57500e1899a0e84a63f0b61ddb28409316b
src/modules/swap-router/SwapRouter.sol	c092a79d9e7cd77d1403a098f0a4af88cf302a874627f13b957c3187392fb811
src/modules/vesting/VestingSimple.sol	a6fed1c606744851537039 34b62962b4312af3c45259599c3981727b679806c8

Contract	SHA256
src/utils/Math.sol	cb399505b6b295199f621ad41a02702ee10e406f36dd37d020473cd94c344fc9
src/utils/SigUtils.sol	d34f2b62e595176746bf032429bff1529a4665fc14a9c953c1508fb354b48e00
src/utils/Uint32Array.sol	1ef52fb12ef961528b14aeb556138932e8f0e9fd681e11bdd872e01df6b25d6d

Note: This document contains an audit solely of the Solidity contracts listed above. Specifically, the audit pertains only to the contracts themselves, and does not pertain to any other programs or scripts, including deployment scripts.

Issue Descriptions and Recommendations

Click on an issue to jump to it, or scroll down to see them all.

~~Q-1~~ `VestingSimple::withdrawAnyFor`'s `Withdraw` event does not conform to comments

~~Q-2~~ Dead link in comments in `Multicall.sol`

~~Q-3~~ Lack of human readable IDs for dependency contracts, in `Registry.sol`

~~Q-4~~ `getCurveV2DerivativeStorage` is an unused mapping in PriceRouter.sol

~~Q-5~~ `Uint32Array.sol` comments are misleading

Q-6 The `constructor` in Cellar.sol does not initialize anything

Q-7 Consider warning exiting LPs that they are leaving guaranteed returns behind during vesting

Security Level Reference

We quantify issues in three parts:

1. The high/medium/low/spec-breaking **impact** of the issue:
    - How bad things can get (for a vulnerability)
    - The significance of an improvement (for a code quality issue)
    - The amount of gas saved (for a gas optimization)
2. The high/medium/low **likelihood** of the issue:
    - How likely is the issue to occur (for a vulnerability)
3. The overall critical/high/medium/low **severity** of the issue.

This third part – the severity level – is a summary of how much consideration the client should give to fixing the issue. We assign severity according to the table of guidelines below:

Severity	Description
(C-x) Critical	We recommend the client **must** fix the issue, no matter what, because not fixing would mean **significant funds/assets WILL be lost.**
(H-x) High	We recommend the client **must** address the issue, no matter what, because not fixing would be very bad, *or* some funds/assets will be lost, *or* the code's behavior is against the provided spec.
(M-x) Medium	We recommend the client to **seriously consider** fixing the issue, as the implications of not fixing the issue are severe enough to impact the project significantly, albiet not in an existential manner.
(L-x) Low	The risk is small, unlikely, or may not relevant to the project in a meaningful way. Whether or not the project wants to develop a fix is up to the goals and needs of the project.
(Q-x) Code Quality	The issue identified does not pose any obvious risk, but fixing could improve overall code quality, on-chain composability, developer ergonomics, or even certain aspects of protocol design.
(I-x) Informational	Warnings and things to keep in mind when operating the protocol. No immediate action required.
(G-x) Gas Optimizations	The presented optimization suggestion would save an amount of gas significant enough, in our opinion, to be worth the development cost of implementing it.

Issue Details

~~H-1~~ `Cellar::_withdrawInOrder` calculations round down due to division before multiplication, which may leave LP's assets in the Cellar

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec	Fixed	High	High

This issue is similar to M-2, in that they both involve dividing, and then using the result from the division to multiply later.

`uint256 totalWithdrawableBalanceInAssets` multiplies `withdrawableBalance` with `exchangeRate`, but `exchangeRate` may lose precision due to Solidity division. This may cause a user to not receive all of the `assets` from the cellar that they are entitled to.

Remediations to Consider

1. Consider using a high internal scaling factor for fixed point numbers so precision loss becomes acceptable
2. Consider making the changes from M-2, to fix the division before multiplication problem in `getValues()` and `getValue()`. Then change `totalWithdrawableBalanceInAssets` in `_withdrawInOrder` to something like:

```
(uint256 exchangeRateNumerator, uint256 exchangeRateDenominator) =
  priceRouter.getExchangeRateFOO(positionAsset, asset);
  // modified version of getExchangeRate that returns numerator and denominator
     instead of dividing, so values can be plugged into equation below, without
     loss of precision.

uint256 totalWithdrawableBalanceInAssets = exchangeRateNumerator *
withdrawableBalance / exchangeRateDenominator / onePositionAsset;
```

With the remediations from M-2 and the `totalWithdrawableBalanceInAssets` change, assets will no longer be erroneously left inside the Cellar.

RESPONSE BY SOMMELIER FINANCE

> `_withdrawInOrder` has been updated to use a scaling factor, the `PRECISION_MULTIPLIER` introduced on line 840 in Cellar.sol. These multipliers are used in the `withdrawInOrder` accounting logic (lines 866-893). This approach also necessitated the creation of a new struct, `WithdrawPricing` (lines 827-835) in order to get around stack-too-deep errors in the fixed implementation.

~~H-2~~ LPs can lose funds to a malicious strategist when calling `redeem`, following a strategist call to `callOnAdaptor` that swaps into an untracked asset

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec	Addressed	High	Medium

While `allowedRebalanceDeviation` puts a limit on the damage, a strategist can submit a call to `callOnAdaptor` that swaps into an asset that is not tracked by the Cellar. If an LP withdraws their shares with `redeem` before the untracked asset issue is found and rectified, the LP would unknowingly leave some of their assets in the `Cellar`.

This can can manifest in a number of ways:

1. Accidentally. The strategist swapped into an untracked positions with no malicious intent.
2. On purpose. A malicious strategist can do the following:

- Create an ERC20, let's call it `AttackCoin`. Only the attacker's EOA holds `AttackCoin`.
- Create a UniswapV3 pool between the Cellar base asset and `AttackCoin` and provide liquidity. There is no risk to the attacker at this stage because no one else holds `AttackCoin`, meaning that no one can drain the base asset liquidity that they provided to the pool. The Attacker doesn't need to provide much liquidity, because they also control the `amountOutMin` argument.
- Call `callOnAdaptor` to swap the base asset for `AttackCoin`. The value will be the max amount allowed through `allowedRebalanceDeviation`.
- Use the EOA to swap `AttackCoin` for the base asset and remove all remaining liquidity from pool.

- Attacker profits, draining funds from the Cellar/LPs. Cellar/LPs are stuck with worthless `AttackCoin`.

As the value in the Cellar rises, it becomes more profitable and attractive to attack. For example, a malicious strategist could take ~ `300_000e18 DAI` out of a `100_000_000e18 DAI` Cellar, given the default `allowedRebalanceDeviation`. This attack can also be repeated by calling `callOnAdaptor` many times, but the rate limiting that will be put in place on the Sommelier chain side should help mitigate its repeatability.

Remediations to Consider

When calling `callOnAdaptor`, consider adding a check that all final out assets are tracked assets in the `Cellar`. **Intermediate swaps of unchecked out assets should still be allowed to allow for greater strategist flexibility.**

RESPONSE BY SOMMELIER FINANCE

> The discussed attack vector is the effect of a deliberate acceptance of design trade-offs when designing an architecture for Cellar adaptors that allows arbitrary function execution (within the functionality of a cellar's already-set-up adaptors).
>
> The design choice here is that the Cellar itself should not have any knowledge about the purpose and/or logic of adaptor calls - the alternative would introduce the need for a complex system of interdependent, end-protocol-aware checks within Cellar logic itself. Therefore, the context-aware suggested remediation is not appropriate for this architecture.
>
> In general, the `totalAssets` check and the `allowedRebalanceDeviation` are the core on-chain guardrails against misappropriation of cellar funds via adaptor calls. Off-chain, the Sommelier chain's Steward architecture has and will continue to build capabilities for detecting, and blocking, malicious-seeming strategist calls. Steward can use a wide range of heuristics not available on-chain, such as "close" approaches to the rebalance deviation, unexpected tokens in decoded parameters, and so on. Given that Steward can detect and block malicious activity as described, the security model basically allows a malicious strategist to get one "free" malicious execution, within the rebalance deviation, before misappropriation is detected and stopped.
>
> Sommelier believes this is an acceptable trade-off of risk vs. system complexity, given that default allowed rebalance deviations have also been reduced significantly in response to this report (see line 1179 of Cellar.sol). As allowed rebalance deviations decrease, and the TVL of a given Cellar increases, it's likely that any single-time exploitation of the rebalance deviation would not be game-theoretically "worth it" compared to continuing to operate a large TVL cellar honestly and earning fees. These off-chain, aligned-incentive considerations could also create room for us increasing allowed rebalance deviations for given trusted cellars, which is a desired feature we may want to employ in the future.

~~H-3~~ Strategists can inadvertently set the `holdingIndex`

TOPIC	STATUS	IMPACT	LIKELIHOOD
Use Cases	Fixed	High	Medium

A strategist can add or remove a position, trusted by the Registry, using `Cellar:addPosition` or `Cellar:removePosition`.

However, this can change what the `holdingIndex` and the holding position (position to deposit to) refer to. `holdingIndex` is a fixed index in the `creditPositions` array.

Downstream impact of confusing the holding position for another arbitrary position:

1. **Withdrawal**: When adjusting the withdrawal preferences for positions using `swapPositions(someIndex, holdingIndex ...)`, a scenario can arise where the intention is to promote the holding position for withdrawal. However, Strategists can inadvertently promote an appreciating position for withdrawal. This can cause a reduction in the appreciating position. In other words, assets are withdrawn in the wrong order, inadvertently lowering the overall value of the Cellar.
2. **Deposit**: When a LP deposits, the Cellar can inadvertently deposit into a depreciating position where the intention was depositing into the `holdingPosition`.

Remediations to Consider

To simplify the implementation, consider defining new state `holdingPosition` (the uint32 ID) instead of `holdingIndex`; this reduces the developer's mental burden by removing the concern of array accesses:

- When mutating the underlying array with `removePosition` consider checking if `creditPositions[toRemoveindex]` refers to `holdingPosition` instead and revert if needed
- Consider removing the code dealing with the holding position in `swapPositions` **because the notion of the holding position is represented as a name instead of an index**

RESPONSE BY SOMMELIER FINANCE

> As suggested, use of `holdingIndex` has been changed to use of `holdingPosition`, with checks modified to check the `positionId` for removal of the holding position (line 239-240 of Cellar.sol), and `swapPositions` code dealing with the holding position removed.

~~H-4~~ `minHealthFactor` is Strategist controlled, potentially making Cellar Aave accounts liquidatable

TOPIC	STATUS	IMPACT	LIKELIHOOD
Trust Model	Fixed	High	Medium

(The important difference between this and H-5 is: H-5 is about the liability side of an Aave account, and this issue about the equity side.)

Strategists can call `addPosition` to add position configuration. A type of position configuration is the `minHealthFactor`. If there is little collateral to liability (low `healthFactor`), the cellar's collateral becomes liquidatable. `minHealthFactor` helps restrict LPs from withdrawing too much collateral from Cellar's Aave account by stopping LPs from withdrawing the aToken positions excessively.

However, the position configuration **does not need to be approved by governance**.

```
// Cellar.sol; addPosition called by Strategist
function addPosition(
  uint32 index,
  uint32 positionId,
  bytes memory configurationData,
  bool inDebtArray
) external onlyOwner whenNotShutdown;

// Registry:trustPosition; called by Governance
getPositionIdToPositionData[positionId] = PositionData({
  adaptor: adaptor,
  isDebt: isDebt,
  adaptorData: adaptorData,
  // Note it is initalized to 0 and
  // to be set in Cellar:addPosition by Strategist
  configurationData: abi.encode(0)
});
```

An issue can arise when:

1. Strategist calls `addPosition` for an `aToken` position with 0.1 as the `minHealthFactor` [1] [4]
2. Strategist deposits collateral into Aave, then borrows assets
3. A LP withdraws from the `aToken` position. The withdrawal causes `healthFactor` to be close to below 1 e.g. 1.0000001 [2][3]
4. The Chainlink oracle price drops for the collateral and the Cellar Aave account becomes liquidatable

Remediations to Consider

Consider making position configuration data configurable by Governance instead of by Strategists to ensure `minHealthFactor` is sufficiently conservative.

Notes

1. Position cannot be withdrawn when value is 0
2. Withdraws cannot directly cause a liquidation because Aave blocks it [error code 6])
3. The difference here is this issue brings down `healthFactor` by withdrawing collateral and H-5 does it by borrowing more
4. This issue can happen regardless of strategist's intention. Strategists can set a dangerous (< 1) `minHealthFactor`, so an excessive withdrawal of Aave collateral can happen

RESPONSE BY SOMMELIER FINANCE

> A constant minimum health factor, `HFMIN`, has been added to lines 71-73 of AaveATokenAdaptor.sol and lines 51-53 of AaveDebtTokenAdaptor.sol. This value is checked against the strategist-set `minHealthFactor` on lines 121-123 of AaveATokenAdaptor.sol and lines 140-142 of AaveDebtTokenAdaptor.sol, putting a hard lower bound on the minimum AAVE health factor and preventing the strategist from liquidating themselves.

~~H-5~~ Strategists can steal by liquidating Aave collateral via sandwich attacking a price oracle update

TOPIC	STATUS	IMPACT	LIKELIHOOD
Trust Model	Addressed	High	Low

The strategist of a cellar can borrow from Aave.

However, this power is unchecked. Strategists can borrow on cellar's behalf at arbitrarily low health factor, provided borrowing does not cause the position to be liquidatable (an Aave restriction, error 11 - "There is not enough collateral to cover a new borrow").

Consequently, strategists are incentivized to sandwich attack around an oracle price drop (e.g. USDC-USD is dropping 1%) because they profit from Aave liquidation fees.

For example, borrow WETH using USDC as collateral

1. Wait for a TX for Chainlink USDC price drop to enter into the mempool
2. Front run TX and borrow WETH to bring health factor lower e.g. 1.02 (below 1 is liquidatable) [1]
3. Chainlink USDC-USD price drop executes
4. Backrun a TX to liquidate collateral

- Health factor was 1.02 before Chainlink's TX
- Suppose USDC price update is -3%, new health factor becomes < 1 after price update tx executes

Remediations to Consider

The root cause is that **strategists can borrow on a cellar's behalf at an arbitrarily low health-factor**.

Consider fetching the health factor from Aave to compare to a configured minimum in `AaveDebtTokenAdaptor:borrowFromAave`.

However, consider using extra caution when designing a solution.

1. Presumably using the same health factor for both the credit and debt adaptor is desirable, for restricting withdrawing too much collateral or borrowing too much. Note that the health factor is configuration for `AaveATokenAdaptor`. Therefore, setting the health factor in 2 different places can lead to inconsistent configuration.
2. In general, a credit and debt adaptor pair for the same protocol are special because they manipulate the same protocol state. Therefore, there can be the same pattern of 2 adaptors

depending on the same configuration in `Cellar`. Consider to be careful with having inconsistent configuration when it comes to a credit and debt adaptor pair.

3. The call from `cellar:callOnAdaptor` to `AaveDebtTokenAdaptor:borrowFromAave` does not allow for passing configuration data. Consider adding a configurable limit for borrowing.

Notes

1. This TX is not the easiest to front run, but it is still possible. There will likely be a very short time window between the Sommelier chain validating Strategist instructions and them getting executed.

2. Because there can be MEV bots watching the Sommelier chain for the signatures for borrowing and frontend the attack on Ethereum.

3. Also because of the fees present in `submitLogicCall`, which will eventually result in bots trying to claim them. In the worst case for Strategists, the Cellar's Aave account will be liquidatable before they can backrun with a liquidation call, leaving the liquidation opportunity to others.

RESPONSE BY SOMMELIER FINANCE

> While we've added a lower bound to the min health factor as described in [H-4], we found no additional action was needed for this scenario, and believe the likelihood to be much lower than the reported "Medium", due to the following mitigating factors:
>
> 1. The hypothetical oracle price drop described in the issue (1%) is double that of Chainlink's widest deviation threshold, which is 0.5%, making the opportunity less profitable.
> 2. The attack requires the strategist/attacker to submit an adaptor call to borrow with extremely precise timing, such that it falls in the same block as the pending oracle update. In practice, strategists have no guarantees about same-block execution - the Sommelier chain needs to perform its own consensus process before submission, as well as Steward validation.
> 3. Since AAVE conducts partial liquidation, the upside of such an attack is limited to the size of the oracle deviation plus the liquidation penalty, and could only be performed once under very specific conditions.

~~M-1~~ `AaveATokenAdaptor:withdraw` confuses LP for Cellar

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec	Fixed	Medium	Low

Cellar calls mutative and trusted adaptor functions via `delegatecall` to change the Cellar's state according to the adaptor's logic. And Cellar uses `staticcall` for the non-mutative functions.

`withdraw` is a mutative adaptor function for the Aave aToken adaptor. `withdraw` fetches the Aave health factor to:

- Compare against a configured minimum
- Decide whether the withdrawal will cause the Cellar's Aave positions to be at risk for liquidation

However, `withdraw` uses `msg.sender` for fetching the health factor meaning `msg.sender` will be the LP because Cellar `delegatecall` s into `withdraw` . **In other words, the health factor of the LP withdrawing will be used instead of the Cellar's.**

Confusing the LP's health factor for the Cellar's health factor can lead to the LP not being able to redeem shares when:

- The LP's health factor is poor (health factor < `minHealthFactor`) but the Cellar's health factor is good.

Remediations to Consider

1. Consider changing `msg.sender` to `address(this)` to correctly represent the cellar's address in `AaveATokenAdaptor:withdraw`
2. Consider defining 2 different helper functions to help with readability for reading Cellar address in `delegatecall` and `staticcall` context to reduce the amount of mental translation needed for the readers

- Note one can leverage Solidity's modifiers and declare `_cellarDelegateCaller` as mutative even it is not, so that using `_cellarDelegateCaller` in a non-mutative context will be a compile-time error

3. `function _cellarDelegateCaller() internal() returns (address);` i.e. `address(this)`
4. `function _cellarStaticCaller() internal view() returns (address);` i.e. `msg(sender)`

References

- https://docs.aave.com/risk/v/aave-v2/asset-risk/risk-parameters#liquidation-threshold

RESPONSE BY SOMMELIER FINANCE

> The `withdrawFromAave` function in AaveATokenAdaptor.sol was updated to use the correct address on line 250.

~~M-2~~ `PriceRouter::getValue()` may return a different result from `PriceRouter::getValues()` due to order of operations and division before multiplication

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec	Fixed	Medium	High

The cause for this discrepancy is `getValue()` and `getValues()` swap the ordering of steps 2 and 3 as outlined below. Multiplication and division are mathematically commutative, but division in Solidity takes the floor, which makes the operation not commutative.

`getValue()` order:

1. Get price in USD of base asset from `_getPriceInUSD()`
2. `getExchangeRate` executes `basePrice.mulDivDown(10**quoteAssetDecimals, quotePrice);`
3. Final amount is `amount.mulDivDown(getExchangeRate(baseAsset, quoteAsset), 10baseAsset.decimals());`

`getValues()` order:

1. Get price in USD of base asset from `_getPriceInUSD()`
2. `_getValues()` executes `(amounts[i].mulDivDown(price, 10**baseAsset.decimals()));`
3. Final amount is `valueInUSD.mulDivDown(10**quoteDecimals, quotePrice)`

The issue is exacerbated as the size of `amount` argument increases.

In `BaseAdaptor::oracleSwap`, `amountIn` relies on `priceRouter.getValue()`, which may artificially push the value down causing otherwise valid swaps to revert on `BaseAdaptor__BadSlippage()`. It can also push `amountIn` up, allowing swaps that should have reverted to succeed.

In `AaveATokenAdaptor::withdrawableFrom`, the `withdrawable` amount depends on `priceRouter.getValue()`, which may result in it being higher or lower than expected.

Remediations to Consider

Consider grouping all multiplication together first, and then doing all division at the end, so no precision is lost in the intermediate steps. For example in `getValues()`:

```
valueInQuote += (amounts[i] * basePrice * 10**quoteDecimals) / 10**baseAsset.decimals() / quotePrice;
```

Also, consider having `getValue()` and `getValues()` follow more similar code paths so they return the same value for the same pricing call. One option is to have `getValue()` construct the proper arguments and call `_getValues()` instead of calling `getExchangeRate()`.

RESPONSE BY SOMMELIER FINANCE

> The Price Router was updated in multiple areas of PriceRouter.sol to use more consistent math and a common internal logic function, `_getValueInQuote`.

M-3 `AaveATokenAdaptor::withdrawableFrom` and `CTokenAdaptor::withdrawableFrom` may lose precision

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec	Acknowledged	Medium	High

This issue is similar to H-1 and M-2. The underlying cause is dividing and using that result to multiply. The divisions in `AaveATokenAdaptor` and `CTokenAdaptor` that are multiplied later in `Cellar::_withdrawInOrder` are the following:

```
// AaveATokenAdaptor::withdrawableFrom
maxBorrowableWithMin =
```

```
    totalCollateralETH - (((minHealthFactor) * totalDebtETH) /
      (currentLiquidationThreshold * 1e14));

  // ...

  uint256 withdrawable = priceRouter.getValue(WETH(),
    maxBorrowableWithMin, underlying);

  // CTokenAdaptor::withdrawableFrom
  return cTokenBalance.mulDivDown(cToken.exchangeRateStored(), 1e18);
```

These divisions can cause `totalWithdrawableBalanceInAssets` in `Cellar.sol` to be lower than expected, resulting in the same scenario as H-1.

Remediations to Consider

1. Consider using a scaling factor outlined in H-1.
2. Consider returning the numerator and denominator from `withdrawableFrom` similarly outlined in H-1.

RESPONSE BY SOMMELIER FINANCE

> The Sommelier team couldn't detect a significant issue in either of the mentioned adaptors, and no code changes were made.
>
> In `AaveATokenAdaptor`, any loss of precision in `withdrawableFrom` will lead to an undercounting, meaning that no unintended liquidations could occur from withdrawing too much, and any amount "stuck" would not be significant.
>
> In `CTokenAdaptor`, we found no way to eliminate the division operation in `withdrawableFrom` without adding a common scaling factor to all adaptor code, which we felt was undesirable due to the "least-knowledge" cellar/adaptor design also described in [H-2].
>
> We found both losses of precision to be extremely insignificant (on the order of needing millions of tokens in order to lose a single base unit to truncation), and not worth fixing given that the suggested remediations result in a degraded architecture.

~~M-4~~ `VestingSimple::deposit` calls `ERC20(asset).transferFrom` without checking return value

TOPIC	STATUS	IMPACT	LIKELIHOOD
Use Cases	Fixed	High	Low

Some `ERC20` tokens return `false` when `transferFrom` fails instead of reverting, which could cause `VestingSimple::deposit` to successfully run even though it didn't receive the tokens it expected.

Remediations to Consider

Consider using `safeTransferFrom` instead of `transferFrom` in `VestingSimple::deposit`.

RESPONSE BY SOMMELIER FINANCE

> This was fixed on line 178 of VestingSimple.sol.

~~M-5~~ Cellar becomes unusable if a UniswapV3 position is added to the Cellar and the second token in the UniV3 pair is not supported by the PriceRouter

TOPIC	STATUS	IMPACT	LIKELIHOOD
Use Cases	Fixed	High	Low

When a position is trusted in the Registry, the Registry checks to make sure the position asset is supported by the PriceRouter.

However, if a UniswapV3 position is added to the Cellar and the second token in the UniV3 pair is not supported by the PriceRouter, no one will be able to call `deposit()`, `withdraw()`, `mint()`, or `redeem()`. This is because these functions need to calculate `totalAssets`, and `totalAssets` is found by finding the balance of every position in the Cellar. When finding the balance of a UniswapV3 position, the price of both the first token and the second token in the UniV3 pair are needed. Since the Registry never checks to make sure the second token in the pair is supported by the PriceRouter, if the PriceRouter does not support the token, these functions cannot be called and will revert.

Remediations to Consider

When trusting a UniswapV3 Position in `registry.trustPosition()`, check to make sure the second token in the UniV3 pair is also supported by the PriceRouter.

Also consider defining a standardized hook to allow Adaptor authors to define custom validation logic.

```
// Registry.sol:trustPosition

// RECOMMENDATION:
// 1. Move logic for "Check that asset of position is supported for pricing operations'
//    It's more coherent this way.
// 2. Optionally, run validation logic custom to the adaptor.
if (!BaseAdaptor(adaptor).isPositionTrustworthy(positionData))
  revert ...
```

RESPONSE BY SOMMELIER FINANCE

> This was fixed by adding an `assetsUsed` field to the common adaptor interface (see base implementation on lines 113-119 of BaseAdaptor.sol). `UniswapV3Adaptor` then overrode the `assetUsed` function to report both LP tokens (lines 181-189), which were stored in the previously-defined `adaptorData`.
>
> Finally, `assetsUsed` was added to the price router support check on lines 274-279 of `Registry.sol`, ensuring that any asset used based on the adaptor's own reporting is supported by the price router.

~~M-6~~ A UniV3 position's underlying worth can be undervalued, causing `totalAssets()` to be undervalued as well

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec	Fixed	Medium	Medium

This issue is similar to H-1 and M-2 in that they both involve dividing, and then using the result from the division to multiply later.

When `totalAssets` of the cellar is calculated and the cellar has UniswapV3 positions, `UniwswapV3Adaptor.balanceOf()` is called to calculate the UniV3 position's underlying worth in terms of the first token in the pair.

However, the amount of `token1` is converted into the amount of `token0` like this:

```
amount1.mulDivDown(price, 10**token1.decimals()
```

`price` is equal to `(basePrice * 10**quoteAssetDecimals) / quotePrice` as seen in `PriceRouter._getExchangeRate()`

So, `amount1` is multiplied by `price` but that multiplication may lose precision due to Solidity division rounding down.

This will cause `totalAssets` to be undervalued. This is seen especially when the `asset` of the cellar has 18 decimals.

Remediations to Consider

Consider changing `getExchangeRate` in `UniswapV3Adaptor.balanceOf()` to something like:

```
(uint256 numerator, uint256 denominator) = PriceRouter(
  Cellar(msg.sender).registry().getAddress(
    PRICE_ROUTER_REGISTRY_SLOT())).getExchangeRate(token1, token0);
```

And then using the `numerator` and `denominator` when returning the balance:

```
return amount0 + (amount1 * numerator) / denominator / (10**token1.decimals());
```

RESPONSE BY SOMMELIER FINANCE

> A scaling factor, `precisionPrice` , was added to the `balanceOf` calculation in UniswapV3Adaptor.sol (lines 93-170). This value is derived from the decimals of the token.

M-7 No mechanism to stop `Cellar` funds from being used to seed a `VestingSimple` contract

TOPIC	STATUS	IMPACT	LIKELIHOOD
Trust Model	Acknowledged	Medium	Low

`Cellar` funds should not be transferred into a `VestingSimple` contract, however that is currently possible. The total amount of damage is limited by `allowedRebalanceDeviation` .

Remediations to Consider

Consider implementing a check to assert that no value is being lost from the `Cellar` and transferred to a `VestingSimple` contract.

RESPONSE BY SOMMELIER FINANCE

> As discussed in [H-2] above, there are myriad ways to misappropriate assets within the `allowedRebalanceDeviation` , and any usage of supported adaptors to perform such an action does not require remediation. The Sommelier team is aware of its chosen security trade-offs and believes the combination of on-chain and off-chain security architecture described in the [H-2] response to be the optimal approach for its use case.

M-8 Strategists can deposit an asset that is not an ERC20 position in the Cellar into a vesting contract

TOPIC	STATUS	IMPACT	LIKELIHOOD
Trust Model	Acknowledged	Medium	Medium

A strategist can transfer any token into the Cellar (bypassing `callOnAdaptor`), and then deposit those assets into the vesting contract, causing `totalAssets()` to increase as vesting occurs. Then, the strategist can withdraw those assets from the vesting contract back into the Cellar after they are vested. Since the asset is not an ERC20 position in the Cellar, `totalAssets()` will decrease, within the rebalance deviation.

LPs that deposited previously will now receive fewer assets than they are owed when they redeem/withdraw. The assets will be left in the Cellar.

Remediations to Consider

Consider asserting that a Vesting contract position had `addPosition` called already on the underlying ERC20. Also consider disallowing `removePosition` calls while that underlying ERC20 is in a Vesting contract.

RESPONSE BY SOMMELIER FINANCE

> Similar to [H-2] above, the suggested remediation does not adhere to the design principles of cellars vis-a-vis their adaptors, and as discussed in both [H-2] and [M-7], any risk from a malicious strategist where the impact is bounded by the rebalance deviation does not require remediation. Congruent to the mentioned issues, the Sommelier team found no required changes here.

~~L-1~~ `VestingSimple::withdrawAll` always emits `Withdraw` event with `amount` as 0

TOPIC	STATUS	IMPACT	LIKELIHOOD
Events	Fixed	Low	High

The `s.vested amount` argument in `emit Withdraw(msg.sender, depositIds[i], s.vested);` is set to 0 a few lines before with `s.vested = 0;`

Remediations to Consider

Consider changing `s.vested` to `shares` when emitting `Withdraw` in `VestingSimple::withdrawAll`.

RESPONSE BY SOMMELIER FINANCE

> This was resolved by storing a `vested` value before resetting the storage slot to 0, on line 238 of VestingSimple.sol, and using the `vested` value for event emission on line 248.

~~L-2~~ If the holding index is out of bounds during `intialize`, no one can deposit into the Cellar

TOPIC	STATUS	IMPACT	LIKELIHOOD
Input Validation	Fixed	Medium	Low

When cellars are initialized in `CellarInitializable.initialize()`, the `holdingIndex` is set. However, there is no check to ensure that the `holdingIndex` is within the range of the credit positions length. Therefore, if `holdingIndex >= _creditPositions.length()`, no one will be able to deposit into the cellar.

Remediations to Consider

In `CellarInitializable.initialize()`, consider checking to make sure that `holdingIndex` is less than `_creditPositions.length()`

RESPONSE BY SOMMELIER FINANCE

> Resolved via use of `holdingPosition`, as described by the response to [H-3].

~~Q-1~~ `VestingSimple::withdrawAnyFor`'s `Withdraw` event does not conform to comments

TOPIC	STATUS	QUALITY IMPACT
Events	Fixed	Medium

The `Withdraw` event is declared as:

```
event Withdraw(address indexed user, uint256 depositId, uint256 amount);
```

The comment on line 29 states that the `user` parameter is the user receiving the deposit.

However, in the `withdrawAnyFor()` `Withdraw` event, the first argument is `msg.sender`, which is the owner of the deposit, not necessarily the receiver of the deposit.

Remediations to Consider

Consider modifying the event to have both the `owner` and the `receiver` as arguments.

RESPONSE BY SOMMELIER FINANCE

> Resolved by adding a fourth parameter to the event signature, for both the `user` (the owner of the deposit) and `receiver` (the address receiving the withdrawal).

> In addition, this parameter separation was added to the deposit event, and both events were renamed to `VestingWithdraw` and `VestingDeposit` respectively, in order to eliminate overlap with the ERC4626-compliant `Deposit` and `Withdraw` events of the cellar.

~~Q-2~~ Dead link in comments in `Multicall.sol`

TOPIC	STATUS	QUALITY IMPACT
Comments	Fixed	Low

Consider changing:

```
* From: https://github.com/Uniswap/v3-periphery/contracts/base/Multicall.sol
```

to

```
* From: https://github.com/Uniswap/v3-periphery/blob/1d69caf0d6c8cfeae9acd1f34ead30018d6e6400/contracts/base/Multicall.sol
```

RESPONSE BY SOMMELIER FINANCE

> The specified link was updated on line 9 of Multicall.sol.

~~Q-3~~ Lack of human readable IDs for dependency contracts, in `Registry.sol`

TOPIC	STATUS	QUALITY IMPACT
Quality	Fixed	Medium

The dependency contracts

1. Gravity Bridge

2. Swap Router

3. Price Router

are assigned **fixed** integer IDs in `Registry.sol`. The dependency Contract-ID relation does not change through the life of the registry. The dependency address can be changed. For example, Swap Router always has registry ID 2, but the Swap Router address (for implementation) can be changed.

However, not all dependency-IDs are named and only the Price Router has a constant human-readable ID - `PRICE_ROUTER_REGISTRY_SLOT = 2` leaving room for readability improvement.

- Consider giving Swap Router and Price Router human-readable IDs to enhance readability. Consider defining immutable names following `_register` calls so there is a lower chance of changing `_register` order and forgetting to change the ID when refactoring:

```
constructor(
  address gravityBridge,
  address swapRouter,
  address priceRouter
) Ownable() {
  register(gravityBridge);
  GRAVITY_BRIDGE_REGISTRY_SLOT = 0; // added

  _register(swapRouter);
  SWAP_ROUTER_REGISTRY_SLOT = 1;    // added

  _register(priceRouter);
  PRICE_ROUTER_REGISTRY_SLOT = 2    // added
}
```

- Consider changing the usage of `registry.getAddress` to use human readable IDs. For example, `registry.getAddress(registry.PRICE_ROUTER_REGISTRY_SLOT())` instead of `PriceRouter(registry.getAddress(2))`.

RESPONSE BY SOMMELIER FINANCE

> All calls to `registry.getAddress` in Cellar.sol were updated to use readable constants instead of integers - lines 497 and 1402.

~~Q-4~~ `getCurveV2DerivativeStorage` is an unused mapping in PriceRouter.sol

TOPIC	STATUS	QUALITY IMPACT
Extra Code	Fixed	Low

In PriceRouter.sol, line 936 declares `getCurveV2DerivativeStorage`.

However, this mapping is never used because whenever a CurveV2 derivative asset is added/set up, the underlying token addresses of the Curve pool are stored in `getCurveDerivativeStorage` instead of `getCurveV2DerivativeStorage`

Consider removing this unused mapping or using `getCurveV2DerivativeStorage` for CurveV2 assets.

RESPONSE BY SOMMELIER FINANCE

> The specified mapping was deleted from PriceRouter.sol.

~~Q-5~~ `Uint32Array.sol` comments are misleading

TOPIC	STATUS	QUALITY IMPACT
Comments	Fixed	Low

In `Uint32Array.sol`, the comments mention `uint256` integers and `uint256[]` arrays.

However, `uint32` integers and `uint32[]` arrays are being used.

Consider switching the comments to reflect the appropriate integer/array.

RESPONSE BY SOMMELIER FINANCE

> All comments were updated to reference `uint32` in Uint32Array.sol.

Q-6 The `constructor` in Cellar.sol does not initialize anything

TOPIC	STATUS	QUALITY IMPACT
Extra Code	Acknowledged	Low

`CellarFactory.deploy` deterministically deploys cellars using Open Zeppelin Clones and initializes the cellars in `CellarInitializable.initialize()` . Therefore, the initializations in the constructor of Cellar.sol are unnecessary. Consider removing them.

RESPONSE BY SOMMELIER FINANCE

> No code changes were made - the Sommelier team desires that Cellar.sol remain a standalone deployable contract, independent from use of the factory. Both factory and direct deployments should be supported.

Q-7 Consider warning exiting LPs that they are leaving guaranteed returns behind during vesting

TOPIC	STATUS	QUALITY IMPACT
Incentive Design	Acknowledged	Medium

Since there is ongoing vesting, an LP is leaving guaranteed returns behind if they exit their position before the vesting is complete. This may not be immediately obvious to LPs, and may incentivize them to stay as LPs for longer.

RESPONSE BY SOMMELIER FINANCE

> The Sommelier team will consider how to address this in user-facing documentation, with no immediate changes made to in-protocol documentation.

> In general, the Sommelier team doesn't see exiting LPs as "losing guaranteed returns" - rather there exists a balance between the vesting returns LPs give up when exiting, vs. the "unearned" vesting returns they receive from previous vesting deposits before their own LP deposit. As such, we don't operate as if there is a conceptual link between the tokens pending in vesting, and any LP funds that may have previously been used to "earn" those tokens.

Disclaimer

Macro makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and Macro specifically disclaims all implied warranties of merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

Macro will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand by any other party. In no event will Macro be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if Macro has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the Emergent team and only the source code Macro notes as being within the scope of Macro's review within this report. This report does not include an audit of the deployment scripts used to deploy the Solidity contracts in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than Macro. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that Macro is not responsible for the content or operation of such websites, and that Macro shall have no liability to your or any other person or entity for the use of third party websites. Macro assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.