macro

Sommelier A-3

Security Audit

October 12, 2022
Version 1.0.0

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 August 15, 2022 to September 2, 2022; and September 26, 2022 to October 7, 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	3	-	-	3
Medium	9	1	1	7
Low	4	-	2	2
Code Quality	11	2	1	8
Informational	10	6	3	1

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:** `d18833d325fa922807f8dc679e81b7b4bcae6606`

Specifically, we audited the following contracts within this repository:

Contract	SHA256
src/Registry.sol	`33717ae5c698beb1c7dea56991ac9ccf552a94727bedc0b7b6203c1c6926f832`
src/base/Cellar.sol	`490b18749f5b3186e7c487a2a1f74c84f2a722be9de8da91a66291935ea4976b`
src/base/ERC4626.sol	`b745b21b01907ace27978063215e5393d08b211c138b50dcd7c44f5d258218a2`
src/base/Multicall.sol	`cbe4ba9c787b262b1b77a83a9969ff9a716d88f09048d175c1a0d00ee4414e14`
src/interfaces/IMulticall.sol	`0d976a972ff63255c5b191b610d7aabe2c43e2918db8e37ee15a28ed3f958586`
src/interfaces/external/IChainlinkAggregator.sol	`bbb3b24935dd23fdd209957901fccc78fd43f876bcd893da81cb31addb155c26`
src/interfaces/external/IGravity.sol	`e09c292aa38c53f6282949761962ef1b8b22f9ea00201477b515a3b44dd37c90`
src/interfaces/external/IUniswapV2Router02.sol	`b1b5ea5db6e598cf9396b64754c4fd2397ff86c6c870b8daf870c5ec427477f6`
src/interfaces/external/IUniswapV3Router.sol	`07e54f895d2e96a922fd26ed7936c3ca432047815f40b9651d613cb73fcb8d81`
src/modules/price-router/PriceRouter.sol	`746f4622db2cd66f37978402dc880d538d312e8c304a7af6d1bab2bc428a0c2e`
src/modules/price-	`dd537dcb2fea34417e26febcab6936a842e76460b77d92437de167f93c3853b6`

Contract	SHA256
router/adaptors/ChainlinkPriceFeedAdaptor.sol	
src/modules/swap-router/SwapRouter.sol	`d2c811ad31a523b80d2e9ead13e72676060298f224bd845f86c458c3bd0fe7e5`
src/utils/AddressArray.sol	`32dd471788ca0b3742177d932c0af3a95fefe0e0c02acde306677e4eb1537e89`
src/utils/Math.sol	`e1b3886632ed1ff5d0498d1f0f8f9d46117e14b579c7b575164731f6b074f5bd`
src/utils/SafeCast.sol	`1a50c67f055d6712749cedd31bd64fa337854978f98157596261d3410f2a4422`

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.

I-2 Cellar position requirements

I-3 ERC4626 position requirements

I-4 ERC20 position requirements

I-5 Performance fee applies to unrealized yield leading to excessive fees for lps

~~I-6~~ `maxWithdraw()` over-estimates by not considering fees

I-7 Governance privilege to add manipulable position is not removable

I-8 Blockchain transparency can create incentives for copying strategy

I-9 Updating to a new PriceRouter may break existing Cellars

I-10 `_previewWithdraw()` misreports for very small asset values

~~Q-1~~ `addAsset()` allows an asset price range to exceed that reported by Chainlink

~~Q-2~~ Incorrect `@notice` comment

~~Q-3~~ Unnecessary event emit

Q-4 Fee-related variable names

~~Q-5~~ Asset existence not checked

~~Q-6~~ Duplicate import

~~Q-7~~ Conflicting libraries

~~Q-8~~ Potential underflow within `withdraw()` and `redeem()`

Q-9 ERC4626 non-conformance with select methods

~~Q-10~~ `AddressArrray.sol`: `add()` will revert due to underflow when the array is empty

Q-11 FAQs not compatible with each other

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~~ Strategists can drain assets of the cellar while doing `rebalance()`. They can manipulate the cellar `totalAssets`, mint more shares than usual, and withdraw later.

TOPIC	STATUS	IMPACT	LIKELIHOOD
Liquidity Pool Manipulation	Fixed	High	Medium

The applied fix limits rebalance damages due to the reported vector. Additional vectors exist (e.g. rebalance-rebalance rebalance-swap) which Sommelier has acknowledged: at this time they elect not to change the contract, but rather utilize Sommelier network mechanics to monitor for occurrences.

A Cellar `rebalance()` operation performs swaps on Uniswap liquidity pools, which incur slippage in relation to the size of the swap and liquidity of the pool. High slippage may result in Cellar `totalAssets` being substantially reduced after swap. A malicious strategist may be able to leverage these conditions to mint more shares per asset value than otherwise possible by orchestrating calls to `rebalance()` intermingled with `deposit()` and `withdraw()`.

Example - Sandwich Attack

Strategist selects a liquidity pool from a pair of Cellar asset positions which has the lowest pool liquidity at present moment. Consider it's an X-Y Pool.

```
1) Rebalance
  - Swap all Cellar assets into one particular asset X
  - Swap all X into Y
  => totalAssets goes down

2) Deposit
  - mint()
  => shares are minted as per reduced totalAssets()

3) Rebalance
  - Swap all Y into X
  => totalAssets are brought back to the initial level.

4) Withdraw
```

```
- withdraw()
=> shares corresponds to more assets than when minted
```

This issue is more severe in the case of UniswapV2 pools since it deploys all liquidity across the whole `xyk` curve. With UniswapV3, the liquidity is concentrated in the normal range, making it less attractive.

RESPONSE BY SOMMELIER FINANCE

> Rebalance protections were added by checking both `totalAssets` and `totalShares` after the execution logic of the rebalance operation. The following checks are:
>
> - `totalShares` must stay constant - since only underlying positions are being swapped, no shares should be created or destroyed.
> - `totalAssets` must stay within a defined deviation range, specified by `allowedRebalanceDeviation`. The allowed rebalance deviation is meant to allow for slippage on the rebalance, while preventing actions that drain accounted-for assets (like swapping to an asset not tracked by a position).
>
> See lines 1512-1517 of `Cellar.sol` for these changes. In addition to the checks, a new function `setRebalanceDeviation` was added in order to edit the defined bounds for acceptable rebalances (lines 1467-1474). This value is designed to be updated by **governance** (not the strategist).

~~H-2~~ Profitable `rebalance` operations can be sandwiched by anyone via `deposit` and `withdraw` for instant profit

TOPIC	STATUS	IMPACT	LIKELIHOOD
MEV Sandwich Attack	Fixed	Medium	High

An MEV attack vector is possible in cases where a rebalance will increase `totalAssets` post-rebalance: Anyone can see a rebalance transaction in mempool and sandwich it between deposit and withdrawal to extract value from the cellar.

Remediations to Consider

Consider delaying reward distribution to resolve this.

RESPONSE BY SOMMELIER FINANCE

> The fixes specified for [H-1] should also preclude MEV attack vectors relating to an increase in totalAssets. See [H-1] for a description of the fixes.

~~H-3~~ Oracle latency allows arbitrage

TOPIC	STATUS	IMPACT	LIKELIHOOD
Price Oracle Arbitrage	Fixed	High	High

A delay in Chainlink price updates can undervalue cellar positions, which creates arbitrage opportunities that can be exploited via sandwich attack for guaranteed profit. This can occur because `Cellar.sol` uses Chainlink to value positions in terms of the Cellar holding asset when pricing share value during `deposit` and `withdraw`:

```
Valuation of positions = sum(holdingPosition-to-position-exchange-rate * position),
```

When an arbitrager observes a price increase transaction queued in the mempool, they can profit by sandwiching the price update with a Cellar `deposit` and `withdraw`. e.g. to get cheap WETH and convert at a higher price post-oracle price update.

Example Sandwich Attack

Initial Conditions:

- Chainlink: WETH-USDC = 100
- Cellar positions: [1MM WETH, 0 USDC (holding position)]
- Cellar share supply: 100MM (@ 1 share/USDC)
- Cellar withdrawType: `ORDERLY`

Attack Block Transactions:

- *[mempool: Chainlink: update WETH-USDC = 101]*
- Arbitrager: **Deposits 100MM USDC**

    - Receives 100MM shares (50% of new total supply)
    - Cellar position: [1MM WETH, 100MM USDC]
    - Cellar total shares: 200MM
- Chainlink: **update WETH-USDC = 101**
- Arbitrager: Redeems 50% shares: **receives 100.5MM USDC** worth of WETH, making 0.5MM USDC worth of WETH risk-free-profit
    - Total assets: (1MM WETH @ WETH-USDC 101) + 100MM USDC = 201MM USDC
    - Redemption: 201MM USDC * 50% = 100.5MM USDC

In the scenario above the cellar acts a market maker on Cellar holdingPositon-position pairs: buying holdingPosition (USDC) and selling position (WETH).

The size is limited by the balance of the position and the buy offer price is set by chainlink oracles. And how much to withdraw from each position depends on withdrawType:

- ORDERLY: the buy offer is buying holdingPosition and selling first (nonholding) position, when first position is depleted then second position and so on.
- PROPORTIONAL is buying holdingPosition and selling a fixed percentage of all positions.

Any deviation between a Chainlink rate and an exchange attracts arbitrager to trade against the Cellar (arbitrager sells holdingPosition, buys position).

Remediations to Consider

Consider requiring the shares to have cliff-vesting. It means the depositor only gets the shares e.g. 10 blocks after depositing.

- An arbitrager will be exposed to the volatility for the positions for 10 blocks. This increases the amount of uncertainty what oracle holdingPosition-position rate will be, thereby increasing uncertainty in arbitrager's expected profit.
- The cost is making `deposit` and `withdraw` not executable in series in the same block for users. Consider finding breaking usecases. It's unlikely there is any as cellars are for retail users that would typically prefer to hold cellar shares for more than 1 block.

RESPONSE BY SOMMELIER FINANCE

> The team implemented a few changes in response to this issue:

- First, a deposit will now lock shares (see line 914 in `afterDeposit` of `Cellar.sol`) for the length of `shareLockPeriod`.
- `shareLockPeriod` defaulted to be locked for 7200 blocks, and can be changed by governance.
- A new function, `_checkIfSharesLocked` (lines 864-870) is checked in `beforeWithdraw` and `_beforeTokenTransfer`, ensuring that shares cannot be transferred or burned within the lock period.
- In order to prevent griefing attacks, where a user can block another user's withdrawals by making small deposits on their behalf and resetting the lock timer, third-party deposit functionality was removed from `CellarRouter.sol`. The Sommelier team could not find a justifiable use case for keeping it, or a workaround that allows third-party deposits yet prevents griefing. We investigated allowing users to set "allowed third-party depositors" but found it not worth the effort.

Note that `_checkIfSharesLocked` needed to leverage `_beforeTokenTransfer` in order to prevent transfers of locked shares: this hook is not implemented in Solmate's ERC20, and would have instead required overriding the `ERC20#transfer` function. This was another determining factor in switching the underlying `ERC20` library from Solmate to OpenZeppelin, as mentioned in the "Code Quality" notes above.

~~M-1~~ ERC4626 vault `maxWithdraw()` logic may allow the vault owner to drain all assets of all Cellars holding it as a position

TOPIC	STATUS	IMPACT	LIKELIHOOD
Spec Exploit	Fixed	High	Low

A malicious third party ERC4626 vault owner can drain all assets of Cellars which hold their vault as a position. This attack can be realized if the vault `maxWithdraw()` logic returns `0` when paused: Cellar `totalAssets()` will exclude the position balance, causing `mint()` to award an increased portion of assets during this time.

See `_balanceOf()` on Cellar.sol line 1475:

```
    function _balanceOf(address position) internal view returns (uint256) {
        PositionType positionType = getPositionType[position];

        if (positionType == PositionType.ERC4626 || positionType == PositionType.Cel
1475:       return ERC4626(position).maxWithdraw(address(this));
```

```
        } else {
            return ERC20(position).balanceOf(address(this));
        }
    }
```

The malicious owner can - via Flash loan - execute a series of `mint()` and `redeem()` / `withdraw()` operations made very favorable by strategically pausing/unpausing their vault, ultimately draining the Cellar of all assets. All of the attack functions are non-privileged. The complexity of executing this vulnerability is variable based Cellar positions, their order, balance ratios, and withdraw type.

Note that a legitimate, ERC4626-conformant vault can be used maliciously by its owner for this purpose: temporarily returning `0` from ERC4626 `maxWithdraw()` is an allowed use-case specifically called out within EIP-4626:

> MUST factor in both global and user-specific limits, like if withdrawals are entirely disabled (even temporarily) it MUST return 0.

Note that this attack vector is not immediately available to malicious Cellar position owners: shutdown is intended to be executable only by governance which makes the Flash Loan attack non-viable.

Related Issues

The root of this issue is `maxWithdraw()` returning a value that may differ from the full balance held. Generalizing on this, there are a number of other potential issues that can occur, which are not necessarily maliciously motivated:

1) Any Cellar may mint too many shares when it holds an ERC4626/Cellar position which under-reports `maxWithdraw()`.

This is a more benign view of the above attack vector. Such under-reporting could be due to non-malicious pausing, staking & locking, or any arbitrary logic which the Vault position owner chooses to implement. Any of these are legitimate and allowed, and arguably the very reason that the `maxWithdraw()` method exists. This scenario grants immediate value to the new minters, which can be extracted when `maxWithdraw()` aligns more closely with balance held (e.g. becomes unpaused; locking period ends, etc).

Consider updating Cellar.sol such that `maxWithdraw()` cannot be overridden in inheriting Cellar logic. This eliminates Cellar positions from this risk category.

2) `redeem()` and `withdraw()` do not revert when all assets cannot be withdrawn, locking assets

Calls to `redeem()` or `withdraw()` can cause a user to receive less `assets` than expected, while still burning `shares` corresponding to the full request. This is most likely to occur when all of the following conditions are true:

- An ERC4626 / Cellar position with a non-zero asset balance returns `0` for `maxWithdraw()`. This may occur if the ERC4626 has logic to return `0` when it is paused/shutdown.
- `withdrawType == PROPORTIONAL` **OR** the amount of `assets` to be withdrawn exceeds the amount available from all other positions

The EIP4626 spec which states for both `withdraw()` and `redeem()`:

> MUST revert if all of `assets` / `shares` cannot be withdrawn (due to withdrawal limit being reached, slippage, the owner not having enough shares, etc).

This shortchanging of value stems from the `_balanceOf()` logic noted above. Subsequently, neither `_withdrawInOrder()` nor `_withdrawInProportion()` validate that the entire `assets` value has been fully withdrawn after cycling through all the positions.

3) Cellars with affected positions may allow minting beyond deposit and liquidity limits

This occurs because `totalAssets()` and `_convertToAssets()` will both return smaller than expected values within `maxDeposit()`, which impacts subsequent calculations to yield a higher than expected values for `leftUntilDepositLimit` and `leftUntilLiquidityLimit` See Cellar.sol lines 1258-1265 of `maxDeposit()`:

```
uint256 _totalAssets = totalAssets();
uint256 ownedAssets = _convertToAssets(balanceOf[receiver], _totalAssets);

uint256 leftUntilDepositLimit = asssetDepositLimit.subMinZero(ownedAssets);
uint256 leftUntilLiquidityLimit = asssetLiquidityLimit.subMinZero(_totalAssets);

// Only return the more relevant of the two.
assets = Math.min(leftUntilDepositLimit, leftUntilLiquidityLimit);
```

Remediations to Consider

Presently `_balanceOf()` returns “withdrawable” data but is used for both “withdrawable” and “holdings” concerns. Consider distinguishing “withdrawable assets” from “held assets”, and using the former within withdraw-related logic. To this end note that Cellar.sol line 1475 could be updated to:

```
return ERC4626(position).convertToAssets(ERC4626(position).balanceOf(address(this)))
```

OR

```
return ERC4626(position).previewRedeem(ERC4626(position).balanceOf(address(this)));
```

If selecting one of these options, consider that `previewRedeem()` would be in conformance with EIP-4626 (`totalAssets()` must be inclusive of fees) but introduce potential slippage to a wider set of Cellar activity, whereas `convertToAssets()` would break EIP-4626 conformance but reduce the scope of slippage concerns to Cellar withdraw activity. Consider documenting your choice and its related considerations.

Also consider extra handling within `redeem()` and `withdraw()` in cases which withdraw-able values do not match balance values. For example:

- There is a shortfall of withdraw-able assets in one or more positions. Consider either reverting with an insightful error, or completing the withdraw from other positions.
- There are not enough withdraw-able assets in all available positions. Consider always reverting with an insightful error.

RESPONSE BY SOMMELIER FINANCE

> The Sommelier team implemented the audit's recommended fixes, adding the new function `_withdrawableFrom` (lines 1754-1761 of `Cellar.sol`), and updating `_balanceOf` to use `previewRedeem` on underlying ERC4626 assets, accounting for the asset's possible discrepancy between `balanceOf` and `maxWithdraw` (change on line 1770). `_getData()` was also updated.
>
> As suggested, `_withdrawableFrom` was leveraged in the cellar's withdraw logic, with both orderly and proportional withdrawals respecting the underlying asset's withdrawal limits (see lines 1126-1138 for orderly withdrawals). Note that in proportional withdraw mode, proportionality is strongly enforced, such that user withdrawal limits are subject to the limitations of the most restrictive underlying assets in terms of withdrawable amounts (line 1186).
>
> Lastly, we implemented a new `maxWithdraw` function overriding the base ERC4626 `maxWithdraw` logic, which accounts for underlying withdrawal limits and current `WithdrawType` settings to report the maximum amount of assets that can be withdrawn at any time (lines 1307-1334). This logic also accounts for fees (lines 1310-1311).

~~M-2~~ Adding a position for an unsupported PriceRouter asset will lock all Cellar assets

TOPIC	STATUS	IMPACT	LIKELIHOOD
Denial of Service	Fixed	High	Low

If a Cellar has a trusted position for an asset that is unsupported by the PriceRouter, the Cellar will become in-operable and all funds locked. Additionally, a new Cellar can be created with an unsupported asset as a trusted position, rendering the Cellar in-operable from the start.

This can occur because the position trust logic within `Cellar.sol` does not validate the asset is supported in `PriceRouter.sol`.

Over time — as the number of Cellars increase and PriceRouter integrations are added — it will become increasingly difficult to manage via off-chain mechanisms that trusted position proposals correspond to supported assets.

Remediations to Consider

Consider adding logic to inspect `PriceRouter.sol` asset support prior to trusting positions in `Cellar.sol`:

- `constructor()` line 715
- `trustPosition()` lines 283-291

If an on-chain asset deprecation notification feature similar to L-2 is implemented, consider reverting when attempting to trust a position whose asset is marked for deprecation.

RESPONSE BY SOMMELIER FINANCE

> The recommended fix was implemented, adding a check on lines 307-308 of Cellar.sol to revert if a newly-trusted asset is not supported by the price router. In addition, logic was added in the constructor to perform the same check (lines 745-747).

~~M-3~~ ETH-USD quote has no circuit-breaker, allowing attackers to arbitrage Cellar assets in extreme market conditions

TOPIC	STATUS	IMPACT	LIKELIHOOD

Price Oracle Arbitrage	Fixed	High	Low

DEFAULT_HEART_BEAT remains at 1 day

If the ETH-USD Chainlink quote were to exceed allowed Chainlink range (presently 1 - 10,000) the ChainlinkPriceFeedAdapter.sol may return stale exchange rate data, allowing attackers to buy-in to Cellar positions are reduced cost.

This can occurs because `_getValueInUSDAndTimestamp()` and `_getPriceRangeInUSD()` disregard the ETH-USD quote `timestamp`; only returning the asset-ETH `timestamp`.

Consider the following scenario, in which 1 Cellar share = 1 asset:

Example baseline conditions (ETH-USD ⇒ 1,000, within Chainlink range):

- asset-ETH quote: 0.01
- ETH-USD quote: 1,000
- The cost in USD of 1,000 assets is 1,000 * 0.01 * 1,000 = 10,000 USD

ETH price sees steep increase (ETH-USD ⇒ 15,000: outside of Chainlink range)

- ETH-USD quote: 10,000 (Chainlink stops recording quotes > 10,000)
- The cost of 1,000 assets:
  - Within Cellars: 1,000 * 0.01 * 10,000 = 100,000 USD
  - In actuality: 1,000 * 0.01 * 15,000 = 150,000 USD
- 1,000 assets mints 1,000 Cellar share, costing 100,000 USD

Users will be able to buy into asset positions at a discount. If the new ETH price is indeed valid Chainlink will deploy Aggregators with an updated, wider range. At that point the assets deposited at a discount within the Cellar can be redeemed with arbitrage profit:

ETH maintains high value, Chainlink updates ranges (ETH-USD ⇒ 15,000: within Chainlink range):

- ETH-USD quote: 15,000
- The cost of 1,000 assets is: 1,000 * 0.01 * 15,000 = 150,000 USD
- 1,000 shares redeems 1,000 assets, valued at 150,000 USD

In this example scenario, an attacker made 150,000 - 100,000 = 50,000 USD in arbitrage.

Remediations to Consider

Consider updating ChainlinkPriceFeedAdaptor.sol to also retrieve the ETH-USD `timestamp`, and return the older of the two `timestamp` values to PriceRouter.sol. This allows circuit-breaker logic to halt Cellar operations under these conditions.

Also consider a shorter default heartbeat value. Presently `DEFAULT_HEART_BEAT = 1 days` creates a window of 1 day during which this arbitrage attack is still possible.

RESPONSE BY SOMMELIER FINANCE

> This issue inspired a deeper refactor of the PriceRouter, such that the logic of ChainlinkPriceFeedAdaptor was integrated into the price router itself. Specific to this report, new logic was added to getValueInUSD , specifying an additional check for staleness on the ETH-USD price if the asset requires a remapping to get its USD price (lines 352-365), and proper conversions if the configured price range is in ETH compared to USD. These checks occur in _checkPriceFeed (lines 429-445).

~~M-4~~ `changeDecimals()` is rounding down to 0 for small values causing a small first mint to break cellar

TOPIC	STATUS	IMPACT	LIKELIHOOD
Denial of Service / Griefing	Fixed	Medium	Medium

`changeDecimals()` in the Math library is used to make conversion between assets with different decimals. This function divides the amount by the difference between decimal numbers if a conversion from an asset with higher decimal to an asset with a lower decimal is requested. However, it is rounding numbers down to 0 if the value is less than 1e^(fromDecimals - toDecimals).

`mint()` calls `_previewMint()` to calculate the assets that will be payed which uses `changeDecimals()`. Because of the issue mentioned above, `changeDecimals()` returns zero for small amounts which causes the mint to be executed without any assets transferred. The requested amount of shares will be minted to the receiver without any payments for the first mint. For the subsequent mints, `_totalAssets` in `_previewMint()` will be zero, causing asset payment amount to be zero as well. This bug will also block deposits and will make the cellar unusable.

The assumption on the L:840 is not correct for the first mint:

```
// No need to check for rounding error, previewMint rounds up.
```

Remediations to Consider

Consider reverting in case it returns zero similar to how it is handled in `deposit()`.

Since `changeDecimals()` is a library function, be sure it works as expected for all use cases.

RESPONSE BY SOMMELIER FINANCE

> As suggested, a revert check was added in the case of a given number of assets rounding down to zero shares after decimal conversion (line 950 of Cellar.sol).

~~M-5~~ If the underlying ERC4626 vault has flash loan capability, an attacker can take a flash loan on the child vault and mint more shares than actual for their deposit

TOPIC	STATUS	IMPACT	LIKELIHOOD
Flash Loan Arbitrage	Fixed	High	Low

In addition to contract fixes, Sommelier indicates off-chain remediation. See response below

Consider there is a Cellar with a position in flash-loan featured ERC4626 vault.

Here attacker can take a flash loan on a given vault and then in a flashloan callback, deposit assets in a cellar. They would get more shares than actual since `totalAssets` for Cellar is temporarily reduced.

Remediations to Consider

Consider doing a thorough analysis of a position before allowing it in `isTrusted`. There shouldn't be a way for anyone to temporarily reduce the `totalAssets` of the underlying vault and then bring it back.

RESPONSE BY SOMMELIER FINANCE

> A level of protection was added around this issue by adding reentrancy guards to `deposit`, `mint`, `withdraw`, and `redeem` functions (preventing looping in order to amplify this attack vector).

> This reported issue also inspired a number of off-chain criteria for cellar positions that will need to be considered and enforced by Sommelier governance:
>
> - Underlying ERC4626 assets should not have flash loan capability
> - Underlying ERC4626 assets should not be upgradeable
>
> This attack vector was also mitigated by adding the share lock feature recommended in reported issue [H-1] of Exponential's crowd audit. Because of this locking feature and reentrancy, users are not able to both deposit and withdraw in a single transaction.

M-6 As cellars can have dynamic structures as underlying positions, methods depending on `totalAssets()` may exceed the gas limit, locking assets.

TOPIC	STATUS	IMPACT	LIKELIHOOD
Gas Limit / Denial of Service	Acknowledged	Medium	Low

A Cellar can invest in other Cellars, which can themselves do the same in turn. Increased complexity of this "investment graph" can cause out-of-gas issues for Cellars higher in the hierarchy, yielding denial of service and locking assets in those Cellars.

Consider a Cellar **X** that holds a position in Cellar **Y**, among other other ERC20/ERC4626 positions. Calling `X.totalAssets()` will cycle through all of it's positions to request their balance held, which includes a call Y. This call to Y will also invoke `Y.totalAssets()`. In this way the entire investment graph will be traversed. Investment additions which extend the graph at lower levels can cause the higher-level Cellars to experience out-of-gas for any calls to `totalAssets()`.

In some conditions the locked Cellar strategist may be able to rebalance out of positions to unlock their Cellar. However, extreme conditions may make this impossible. In those cases, the only means to unlock the Cellar would be to have the other Cellar lower in the hierarchy rebalance out of positions.

RESPONSE BY SOMMELIER FINANCE

> Related to [M-5], the Sommelier team is developing a more robust framework about the different types of assets a cellar can hold. In addition to the restrictions above, we are planning to limit the level of "cellar nesting" that can occur, such that:

- A cellar (parent cellar) can have another cellar (child cellar) as a position, but the child cellar **cannot** have a third cellar as a position.

It's worth noting that there are many reasons it is difficult to place strict upper bounds on `totalAssets` gas consumption:

- Cellars can have a wide range in terms of the *number* of positions they hold
- Cellars with non-ERC20 positions (i.e. ERC4626) are subject to the gas consumption of the underlying position's own contract logic
- Each position must be priced by the `PriceRouter`, which itself can consume more or less gas compared to how simple or complex it is to price the asset (for instance, an ERC20 asset is simple to price, but a DEX LP token requires logic to split the token into its component assets, and price them individually, consuming more gas). A mitigating factor here is that unlike the preceding bullet point, we control the logic of the price router.

Overall, [M-5] and [M-6] inspire a range of considerations and design restrictions that the Sommelier team will consider, and in the future formalize, in future cellar design and governance operations.

M-7 Price oracle with no redundancy can lead to arbitrage

TOPIC	STATUS	IMPACT	LIKELIHOOD
Price Oracle Arbitrage	Wont Do	High	Low

A Cellar depends upon Chainlink to retrieve asset values in USD, and is therefore susceptible to potential attacks from within the Chainlink network.

A chainlink price feed is:

- a proxy that relays price requests to an aggregator
- an aggregator caches the medianized rate aggregated from offchain oracles

Chainlink docs on price feed overview:

> "Using proxies enables the underlying aggregator to be upgraded without any service interruption to consuming contracts."

However, there are multiple attack vectors:

1. Proxies are owned: The 4-9 multisig owner controlled by Chainlink can potentially upgrade the aggregator the proxy points to to a malicious one.
    - A malicious aggregator can then report a rate close to the typical rate so the attack doesn't trip dapps' circuit breakers and still over/undervalue an asset.
    - The protection from min and max price checks will have room for malicious price manipulation within volitile assets such as ETH-USD (+/- 20% in one day).
2. Offchain oracles can collude and report a malicious rate, without needing to change aggregator.

Consider using a secondary oracle such as uniswap or tellor, then require the 2 rates be in sync (e.g. < 2% difference).

RESPONSE BY SOMMELIER FINANCE

> Price feed redundancy will be an ongoing effort for Sommelier cellars, but at this time we could not find a better alternative to Chainlink, and adding new redundancy mechanisms comes with increased points of failure, integration effort, and gas costs per transactions.
>
> So, currently, the protocol relies on an assumption of trust in Chainlink oracles. In the future, we will investigate adding Tellor as an oracle, adding circuit breakers in case of suspicious Chainlink activity, and watch the oracle landscape for new providers as robust and established as Chainlink in terms of both on-chain integration points and operational practices.

~~M-8~~ `_convertToFees()` underflow leading to denial of service by force feeding

TOPIC	STATUS	IMPACT	LIKELIHOOD
Denial of Service / Griefing	Fixed	High	Low

While `totalShares` = 0 an attacker can force feed tokens into one of the Cellar positions to cause `withdraw()`, `redeem()`, `deposit()`, `mint()` and `sendFees()` to revert, creating denial-of-service conditions.

This occurs because `totalAssets()` and hence `feesInShares` will be non-zero, causing `_convertToFees()` to revert due to underflow when calculating `denominator`:

```
    function _convertToFees(uint256 feesInShares) internal view returns (uint256 fee
        uint256 totalShares = totalSupply;
        uint256 denominator = totalShares - feesInShares;
        ...
```

The DoS can be recovered from by executing a rebalance operation.

Remediations to Consider

Consider updating `_convertToFees()` logic to account for conditions of underflow.

RESPONSE BY SOMMELIER FINANCE

> The suggested fix was implemented, with fees only being calculated when totalShares > feesInShares. See line 1645-1650 of Cellar.sol. This should prevent denial-of-service due to underflow in _convertToFees.

~~M-9~~ Underlying Cellar `asset` is mutable, which may allow Cellar value manipulations

TOPIC	STATUS	IMPACT	LIKELIHOOD
Variable Mutability	Fixed	High	Low

The solmate implementation of ERC4626 has the `asset` variable as immutable; whereas in base/ERC4626.sol:33 `asset` is no longer immutable. A mutable value allows inheriting Cellar implementations to potentially change the `asset`, which may cause a variety of issues including the inability to deposit into a cellar or lack of update to the `totalAssets()` valuation.

Remediations to Consider

Consider updating `asset` to be immutable.

RESPONSE BY SOMMELIER FINANCE

> The Cellar's asset will never change after deployment, so that should be immutable.

Reversed PriceRouter asset `minPrice` and `maxPrice` values can break Cellars

TOPIC	STATUS	IMPACT	LIKELIHOOD
Denial of Service / Validation	Fixed	Medium	Low

`addAsset()` will accept `minPrice` and `maxPrice` arguments with reversed values (min > max). Should this occur, all asset price inquiries will always revert, causing the majority of public Cellar.sol methods to always revert: `deposit()`, `mint()`, `withdraw()`, `redeem()`, `sendFees()`, `resetHighWatermak()`, `totalAssets()`, `convertTo*()`, and `preview*()`.

This can occur because `addAsset()` will not perform validations on `minPrice` or `maxPrice` if they are both nonzero. See PriceRouter.sol, line 91-120:

```
if (minPrice == 0 || maxPrice == 0) {
    ... validation logic ...
}

getAssetConfig[asset] = AssetConfig({
    ...
    minPrice: minPrice,
    maxPrice: maxPrice,
    ...
```

Price inquiries funnel through `_getValueInUSD()` within `PriceRouter.sol` lines 301-305 which contains checks against the configured min/max price values:

```
uint256 minPrice = config.minPrice;
if (value < minPrice) revert PriceRouter__AssetBelowMinPrice(address(asset), value,

uint256 maxPrice = config.maxPrice;
if (value > maxPrice) revert PriceRouter__AssetAboveMaxPrice(address(asset), value,
```

If `minPrice` > `maxPrice`, any `value` amount will trigger one of these reverts.

Affected Cellars can be unlocked by calling `addAsset()` with corrected values for `minPrice` and `maxPrice`.

Remediations to Consider

Consider updating `addAsset()` to also validate `minPrice` and `maxPrice` when they are both nonzero.

RESPONSE BY SOMMELIER FINANCE

> An additional check was added in the price router's addAsset function, in order to revert if minPrice >= maxPrice (line 132 of PriceRouter.sol).

~~L-2~~ Removing a PriceRouter asset may lock all Cellar assets

TOPIC	STATUS	IMPACT	LIKELIHOOD
Locked Assets	Fixed	Medium	Low

This is the alternate side of M-2. Whereas M-2 is concerned with a Cellar adding a position for an asset unsupported by the PriceRouter, this issue is concerned with the PriceRouter removing an asset which Cellars already have a position in.

Specifically, when `removeAsset()` is called for an asset which Cellars have an existing position in, all assets in those Cellars will become locked. This occurs because `PriceRouter.sol` `_getValueInUSD()` will now revert on line 294:

```
if (!config.isSupported) revert PriceRouter__UnsupportedAsset(address(asset));
```

`_getValueInUSD()` is at the root of many Cellar.sol methods. Most notable for an invested Cellar user are `redeem()` and `withdraw()`. These methods will revert, locking all invested funds. Affected Cellars can be unlocked by re-adding the asset.

Remediations to Consider

Consider an on-chain and/or off-chain notification scheme for PriceRouter assets which will become deprecated, allowing Cellar Strategists appropriate time to rebalance out of the asset.

RESPONSE BY SOMMELIER FINANCE

> Given the finding, the Sommelier team could not find a justifiable use case for removing an asset from the price router, given the possible downstream effects of removing an asset, and the fact that a price router should not "know" what assets cellars are using. Therefore, all asset removal logic was removed.

L-3 `_withdrawInOrder()` may drain positions that come before the holdingPosition which will require frequent rebalances

TOPIC	STATUS	IMPACT	LIKELIHOOD
Use Case	Wont Do	Low	Medium

There is no mechanism enforcing the first element of the `positions` array to be `holdingPosition` . This will cause `_withdrawInOrder()` to empty positions with an index smaller than `holdingPosition` and may require frequent rebalances to keep the strategy as intended.

Remediations to Consider

Consider requiring index of `holdingPosition` to be zero in constructor and `setHoldingPosition()`

RESPONSE BY SOMMELIER FINANCE

> The Sommelier team considers this the responsibility of the strategist - the strategist should make sure that any configured withdrawal order's implications on the holding position is considered. As such, no code changes were made. We thought it best to maintain flexibility and could see use cases where e.g. a strategist wanted to use anticipated withdrawals, and a defined withdrawal order, as a way of gradually rebalancing the cellar.

L-4 Wrongly assigned position types will block withdraws

TOPIC	STATUS	IMPACT	LIKELIHOOD
Locked Funds / Validation	Wont Do	Low	Low

If a wrong `PositionType` is assigned for a position in `constructor()` or `trustPosition()` , withdraws will be blocked. This situation can be remedied by calling `trustPosition()` with correct type but since it is a governance only function it will take time to vote and execute.

Remediations to Consider

- Detecting the types of positions by making a view call to the target address instead of manually passing PositionTypes.

- If manual type registration is preferred, it is possible to check if the passed type is correct using the technique mentioned above.

RESPONSE BY SOMMELIER FINANCE

> Given the multiple layers of review as part of Sommelier governance, the team prefers manual registration and considers the likelihood of this impact to be small. If it were to occur, as mentioned in the report, it is recoverable by calling `trustPosition()` again with the right position type.

I-1 Cellar network approval

TOPIC	STATUS	IMPACT
Governance	Acknowledged	Informational ✳

Cellar.sol implements many important baseline requirements which appear to be critical to normalized good behavior within the Sommelier network. When adding a new Cellar to the Sommelier network, consider enforcing these requirements via appropriate mechanisms:

- Cellars must inherit from Cellar.sol
- Cellars should not be upgradeable
- Cellars must not lock withdrawals (referencing M-1)

RESPONSE BY SOMMELIER FINANCE

> All informational suggestions have been read and studied and will be the foundation for a set of formalized “cellar requirements” as we move to Cellars V2 and wider community development of cellars (also see discussion of such in [M-5] and [M-6]).
>
> In the meantime, we work closely enough with the current governance validator set, and the current strategists, such that have a high level of confidence of being able to enforce these considerations before they become formalized.

I-2 Cellar position requirements

TOPIC	STATUS	IMPACT
Governance	Acknowledged	Informational ✳

To further ensure that only truly trusted Cellars are operated within Sommelier, consider enforcing that Cellar positions must have also been explicitly approved on the network (referencing I-1).

For additional on-chain trust, consider implementing a central repository contract to act as a source of truth for all Ethereum-wide trusted Cellar addresses. Such a repository could be managed via appropriate approval/control mechanisms. Consider integrating this registry into the “trusted position” Cellar.sol logic so un-approved Cellars cannot be trusted/added as positions.

RESPONSE BY SOMMELIER FINANCE

> All informational suggestions have been read and studied and will be the foundation for a set of formalized “cellar requirements” as we move to Cellars V2 and wider community development of cellars (also see discussion of such in [M-5] and [M-6]).
>
> In the meantime, we work closely enough with the current governance validator set, and the current strategists, such that have a high level of confidence of being able to enforce these considerations before they become formalized.

I-3 ERC4626 position requirements

TOPIC	STATUS	IMPACT
Governance	Acknowledged	Informational ✳

Consider enforcing via appropriate mechanisms:

- ERC4626 positions should not be upgradeable
- ERC4626 positions must not lock withdrawals (referencing M-1)
- ERC4626 positions must not have Flash Loan capabilities (referencing M-5)

RESPONSE BY SOMMELIER FINANCE

> All informational suggestions have been read and studied and will be the foundation for a set of formalized “cellar requirements” as we move to Cellars V2 and wider community development of cellars (also see discussion of such in [M-5] and [M-6]).
>
> In the meantime, we work closely enough with the current governance validator set, and the current strategists, such that have a high level of confidence of being able to enforce these considerations before they become formalized.

I-4 ERC20 position requirements

TOPIC	STATUS	IMPACT
Governance	Acknowledged	Informational ✳

Consider enforcing via appropriate mechanisms:

- ERC20 should not have a fee on transfer
- ERC20 should not be a rebasing token.
- ERC20 must not be a double-sided token (Synthetix’s Tokens)

RESPONSE BY SOMMELIER FINANCE

> All informational suggestions have been read and studied and will be the foundation for a set of formalized “cellar requirements” as we move to Cellars V2 and wider community development of cellars (also see discussion of such in [M-5] and [M-6]).
>
> In the meantime, we work closely enough with the current governance validator set, and the current strategists, such that have a high level of confidence of being able to enforce these considerations before they become formalized.

I-5 Performance fee applies to unrealized yield leading to excessive fees for lps

TOPIC	STATUS	IMPACT

Fees Calculation	Wont Do	Informational ✳

The following functions take performance fee shares when yield has accrued: `deposit()`, `mint()`, `withdraw()`, `redeem()`, `sendFees()`.

Performance fees are taken on the yield of the entire cellar (highwater mark in relation to `totalAssets`), instead of only on realized yield (highwatermark in relation to individual user investment).

Consider the following example:

- Block T: Current cellar asset value is 100MM USDC, 100MM USDC is the high watermark.
- Block T+1: Current cellar asset value increased to 300MM USDC, a LP withdraws a small amount: 1 USDC. The result is Cellar mints 7% performance fee shares because 0.1 * 200MM / 300MM ~= 7% (performance fee rate * yield / total asset).
- Block T+2: Current cellar asset value returns to 100MM USDC, each LP owns 7% less asset due to one LP decides to withdraw.

RESPONSE BY SOMMELIER FINANCE

> The Sommelier team is aware of this design decision as regards the way fees accrue to strategists. Future cellar architecture updates will include revamped fee structuring.

~~I-6~~ `maxWithdraw()` over-estimates by not considering fees

TOPIC	STATUS	IMPACT
Spec Compliance	Addressed	Informational ✳

Per the `maxWithdraw` ERC4626 spec:

> MUST return the maximum amount of assets that could be transferred from owner through withdraw and not cause a revert

However, the present `maxWithdraw` implementation does not account for the diluting effect of performance shares, therefore over-estimate withdrawable asset, and causing

`withdraw(maxWithdraw())` to revert.

I-7 Governance privilege to add manipulable position is not removable

TOPIC	STATUS	IMPACT
Governance	Wont Do	Informational ✳

From Sommelier docs:

> "In general, when designing cellar contracts, one must always minimize trust."

Current implementation allows Governance to maintain an allowlist of assets Strategist can rebalance to/from. This prevents Strategist from adding low liquidity asset and subsequently manipulate the price to extract other Cellar assets. However, Governance still has the same privilege.

While it may make sense from development perspective for Governance to maintain `positions` allowlist, there is no way to remove that privilege after Cellar contracts is deployed.

Once Governance is confident in the assets in the allowlist, it can improve trust for LPs by ossifying the allowlist.

Consider to use a role to maintain the list and revoke the role when Governance deems appropriate. And consider using Open Zeppelin's `AccessControl.sol`.

RESPONSE BY SOMMELIER FINANCE

> The Sommelier protocol, both within cellars and for the Gravity bridge, is currently tightly-coupled to the chain security of the Sommelier chain. If validators were able to collude to destroy the BFT trust properties of the Sommelier chain, all cellars would be vulnerable.
>
> In the future, we may investigate ways to revoke governance permission to update `positions`, but governance would still have control over things like the `Registry` and `PriceRouter`, meaning that a set of colluding validators would have a range of other threat vectors to employ.

I-8 Blockchain transparency can create incentives for copying strategy

TOPIC	STATUS	IMPACT
Use Case	Acknowledged	Informational ✳

A successful cellar's strategy, in particular its `rebalance` operations, can be mimicked perfectly regardless how private the offchain computation is. Because all `rebalance`s and position balances are public. There can be an incentive for LPs to start a clone Cellar to save performance and platform fees. One can set up an off-chain event subscriber and onchain contract to mimic the cellar's `rebalance` i.e. the trades, as well as copy the cellar's strategy to offer lower fees, different incentives, trust policy to attract LPs etc. These cloned trades can even frontrun the cellar's original trades.

RESPONSE BY SOMMELIER FINANCE

> Acknowledged as a basic property of on-chain state transparency in the EVM. It's important to note that while actions/rebalances could be copied, the underlying strategy and the data feeds that inform the strategy remain private to Sommelier validators.
>
> In the future, Sommelier is interested in the evolving zero-knowledge space as a way to propagate updates from strategists across the Gravity bridge, without revealing the content of those updates. Given that research and development in this area is still nascent, we do not expect the ability to copy cellar actions to be restricted any time in the near future.

I-9 Updating to a new PriceRouter may break existing Cellars

TOPIC	STATUS	IMPACT
Denial of Service	Acknowledged	Informational ✳

If the `priceRouter` address is changed in `Registry.sol`, a Cellar may have a trusted position for an asset that is unsupported by the new PriceRouter. This could cause a Cellar to become in-operable and its funds locked.

RESPONSE BY SOMMELIER FINANCE

> More than likely we won't need to update the price router. If we do, it will involve extensive integration testing to insure none of our cellars stop working.

I-10 `_previewWithdraw()` misreports for very small asset values

TOPIC	STATUS	IMPACT
Math	Wont Do	Informational ✳

`_previewWithdraw()` is susceptible to similar siphoning of small values as found in M-4, under the following conditions:

- `totalShares` == 0 and `totalAssets()` != 0. These conditions require that:
  - an asset was directly transferred to the Cellar (not via `deposit()` / `mint()`)
  - `deposit()` / `mint()` have not been called yet
- The asset has decimals > 18

~~Q-1~~ `addAsset()` allows an asset price range to exceed that reported by Chainlink

TOPIC	STATUS	QUALITY IMPACT
Validation	Fixed	Low

An asset can be successfully added with `minPrice` and `maxPrice` outside the valid range that Chainlink reports. This may result in the PriceRouter's price buffering logic no longer taking effect for the asset. This occurs because the `addAsset()` method only performs Chainlink validations on `minPrice` and `maxPrice` if at least one of them is zero. If they are both non-zero, Chainlink price range is not referenced.

Consider the following:

- Chainlink's asset range: 100 - 10,000

- PriceRouter correctly buffered asset range: 110 - 9,000
- PriceRouter incorrectly configured asset range: 1 - 100,000

In the "incorrectly configured" scenario, the PriceRouter will no longer revert for Chainlink values which fall within the 10% buffered zones: 100-110 or 9,000-10,000. As a result, Cellar activity can proceed while the asset value is within the % buffered zones, which would have otherwise reverted. Asset values within this buffered zone are still valid Chainlink price quotes so there is no hard impacts per-se, except a misalignment with expected business logic.

Remediations to Consider

Consider updating logic to also validate `addAsset()` `minPrice` and `maxPrice` arguments against Chainlink if both are nonzero.

RESPONSE BY SOMMELIER FINANCE

> Implemented. PriceRouter.sol lines 120-130

~~Q-2~~ Incorrect `@notice` comment

TOPIC	STATUS	QUALITY IMPACT
Documentation	Fixed	Low

The `@notice` value on Cellar.sol line 101 appears to be incorrect: suspect copy-paste from prior error comments.

RESPONSE BY SOMMELIER FINANCE

> Fixed. Cellar.sol line 102

~~Q-3~~ Unnecessary event emit

TOPIC	STATUS	QUALITY IMPACT

Events	Fixed	Low

The `ShutdownChanged` event can be emitted when no change has occurred, which may cause confusion to users. Consider reverting if `liftShutdown()` is called when the Cellar is not shutdown.

RESPONSE BY SOMMELIER FINANCE

> Fixed. Cellar.sol line 102

Q-4 Fee-related variable names

TOPIC	STATUS	QUALITY IMPACT
Variable Naming	Wont Do	Low

Variables within Cellar.sol `_takePerformanceFees()` are dealing strictly with performance fees, but are named as platform fees. See lines 1340-1347, in particular `feeInAssets` and `platformFeesInShares`:

```
uint256 feeInAssets = _previewPerformanceFees(_totalAssets);
if (feeInAssets > 0) {
    uint256 platformFeesInShares = _convertToFees(_convertToShares(feeInAssets, _tot
    if (platformFeesInShares > 0) {
        feeData.highWatermark = _totalAssets;
        _mint(address(this), platformFeesInShares);
    }
}
```

Consider renaming these variables `performanceFeeInAssets` and `performanceFeeInShares` respectively.

~~Q-5~~ Asset existence not checked

TOPIC	STATUS	QUALITY IMPACT
Validation	Fixed	Low

PriceRouter.sol:L130 `removeAsset()` does not check if that asset actually exists.

RESPONSE BY SOMMELIER FINANCE

> No longer relevant - removeAsset has been removed.

~~Q-6~~ Duplicate import

TOPIC	STATUS	QUALITY IMPACT
Duplicate Code	Fixed	Low

SwapRouter.sol:L9 Multicall is imported twice

~~Q-7~~ Conflicting libraries

TOPIC	STATUS	QUALITY IMPACT
Code Consistency	Fixed	Low

Two different versions of SafeCast are used in the project:

- Cellar.sol:L8
- ChainlinkPriceFeedAdaptor.sol:L10

RESPONSE BY SOMMELIER FINANCE

> Addressed by always using the OZ version of SafeCast. We deleted our handrolled version.

~~Q-8~~ Potential underflow within `withdraw()` and `redeem()`

TOPIC	STATUS	QUALITY IMPACT
Underflow	Fixed	Low

Cellar.sol:L889 & L936 `withdraw()` and `redeem()` don't check if the allowed amount is bigger than shares which is causing underflow errors. Consider checking if allowed > shares and returning a meaningful error.

RESPONSE BY SOMMELIER FINANCE

> Fixed, using `_spendAllowance` from OZ ERC20 (lines 994-998 in Cellar.sol). Note that our underlying ERC20 implementation changed from Solmate to OZ.

Q-9 ERC4626 non-conformance with select methods

TOPIC	STATUS	QUALITY IMPACT
Spec	Acknowledged	Low

ERC4626 specification states the following functions must not revert: `totalAssets()`, `maxDeposit()`, `maxWithdraw()`, `maxMint()`, `maxReeem()`. However, `totalAssets()` -- which the other named functions depend upon -- can revert due to price oracle logic.

Consider documenting this non-conformance for integrators.

RESPONSE BY SOMMELIER FINANCE

> Acknowledged and documented that we depart from the ERC4626 standard here. See line 1202 of Cellar.sol.

~~Q-10~~ `AddressArrray.sol` : `add()` will revert due to underflow when the array is empty

TOPIC	STATUS	QUALITY IMPACT
Underflow	Fixed	Low

Within `add()`, `len - 1` will underflow when `array.length` = 0:

```
function add(
        address[] storage array,
        uint256 index,
        address value
    ) internal {
        uint256 len = array.length;
        array.push(array[len - 1]);
        ...
```

This is not causing issues within the present Cellar.sol implementation: the Cellar `constructor()` initializes `AddressArray` values without using `add()`; and subsequent conditions prevent the array from becoming empty again.

Remediations to Consider

Consider correcting this logic to account for a 0-length array.

Q-11 FAQs not compatible with each other

TOPIC	STATUS	QUALITY IMPACT
Documentation	Acknowledged	Low

From FAQs on the landing page

> Can strategy providers add new positions to the Cellar?

> Although strategy providers can change the strategy logic (algorithm which sends position signals to the Cellar), they must submit position additions to Sommelier Governance. This adds a layer of safety for users who have allocated funds to Cellars. They can rest easy knowing that their assets can only be in the prescribed positions that each Cellar manages.
>
> Can strategy providers add new DeFi protocols?
>
> No, for the same strategy provider to add new DeFi protocols new Cellar should be created.

The 2 Q&As are not compatible. Consider to update the documentation for the latter Q&A, to reflect Strategist **can** add a new position, provided Governance approves the addition.

RESPONSE BY SOMMELIER FINANCE

> This copy update has been shared with the relevant teams.

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.