macro

Sommelier A-12

Security Audit

October 27, 2023
Version 1.0.0

Presented by 0xMacro

Table of Contents

Introduction

This document includes the results of the security audit for Sommelier's smart contract code as found in the section titled ‘Source Code’. The security audit was performed by the Macro security team from October 24, 2023 to October 26, 2023.

The purpose of this audit is to review the source code of certain Sommelier 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
Medium	2	-	-	2
Low	2	-	-	2

Sommelier was quick to respond to these issues.

Specification

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

- Discussions with the Sommelier team.
- Available documentation in the repository.

Trust Model, Assumptions, and Accepted Risks (TMAAR)

Trusted entities:

- Strategists:
    - User that can manage positions in the cellar. Is trusted to act in the benefit of the cellars shareholders, and earns a portion of the cellars profits. All actions made by the strategist are approved by Sommelier governance. Has the ability to shutdown the cellar in case of an emergency.
- Governance:

    - Sommelier governance responsible for approving strategist actions, as well as adding or removing trusted positions for cellars.

- Multisig:
    - Approves adaptors and positions in the registry as well as adding and updating price feeds for assets in the priceRouter. Can pause cellars, which can be unpaused by governance.

- Chainlink:
    - Responsible for a majority of pricing, as well as running automated tasks for share price oracles. It is trusted that the data it provides is correct.

The goal of the system is to to have checks and balances for each permissioned action, where if any one permissioned entity acts malicious, the others can remedy the situation, requiring multiple points failure before it can negatively impact users.

Assumptions:

- There is an assumption that permissioned entities will not act maliciously.
- It is assumed that the protocols that a cellar interacts with wont act maliciously, and will operate as intended.

Accepted Risks:

- Share price varies based on market conditions, and there is no guarantee share price will increase.
- Protocols that a cellar interacts with could be exploited. There are ways trusted entities can help mitigate the effect of such exploits, but there may be a negative effect on share price and a loss of funds for users.

Source Code

The following source code was reviewed during the audit:

- **Repository:** cellar-contracts
- **Commit Hash (initial):** `4faacd442fefe1d4d2b39182ad7ed2bc67786d6b`
- **Commit Hash (final):** `4ee541dfd3b494c7eaabd047c73750242f27fe09`

Specifically, we audited the following contracts within this repository for the initial part of the audit, additionally sFrax was evaluated to ensure it would work as intended as a postion using the CellarAdaptor.

Contract	SHA256
src/modules/adaptors/Aura/AuraERC4626Adaptor.sol	`06476b6468024206eeec29752f0ba4b6fa1ac72b0a3aba811c38c27fdde6bc82`
src/modules/price-router/Extensions/Redstone/RedstoneEthPriceFeedExtension.sol	`0968c0307e35690b26dea9586313349a78a16a4528212cb2a2a828c0f2deeaa6`
src/modules/price-router/Extensions/Curve/CurveEMAExtension.sol	`beeb61fd5a2715912f77d80b97192b2075dc84eec8e461acf889a16ae2eff69c`
src/modules/adaptors/Frax/CollateralFTokenAdaptor.sol	`25b8314ebfd899449ec4f5b08535d08307494aaa1058e63112d4a7258ff025f0`
src/modules/adaptors/Frax/CollateralFTokenAdaptorV1.sol	`c4f9e9a1f768e3074a5dbc8b8f37bb3ae1389d6331dccb7e8f6d06b53ce6a38d`
src/modules/adaptors/Frax/DebtFTokenAdaptor.sol	`61fe645c1b296b781246c5864a08f74337ff9d5fc95501de9c8fd9689e8fc953`

Contract	SHA256
src/modules/adaptors/Frax/DebtFTokenAdaptorV1.sol	e702784c116a8d5cf1460e1e38ca00c34656ca02768dcadb52ecd068cc1b7133
src/modules/adaptors/Frax/FTokenAdaptor.sol	94c03365dfa57e6f9a6f377e2eb94d34c7132e961a8f563b840cfbe64138bf09
src/modules/adaptors/Frax/FTokenAdaptorV1.sol	c3a2638038d5526d13882288c69a0e56ddc4292026348ab952d11b9d7f09895f
src/base/ERC4626SharePriceOracle.sol	998cc4bd75ed70f6c6fd15b6fc33ec50547dc26435d54ff0fcdec9486fb730cd
src/modules/adaptors/ERC4626Adaptor.sol	3675d603b20d0c6bc7a76d0b822badda437616d9d3b9a1df341ef612c82da2fe

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.

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

~~M-1~~ Incorrect price calculation for `CurveEmaExtension`

TOPIC	STATUS	IMPACT	LIKELIHOOD
Protocol Design	Fixed	Medium	Medium

In `CurveEmaExtension.sol`'s `getPriceInUSD()`, the price is pulled from the pool in terms of the first asset in the pools `coins` array. When the price is converted to `USD`, the price's decimals are assumed to be using the `coins0` 's decimals.

```
function getPriceInUSD(ERC20 asset) external view override returns (uint256 pri
    ExtensionStorage memory stor = extensionStorage[asset];
    CurvePool pool = CurvePool(stor.pool);

    ERC20 coins0 = getCoinsZero(pool);
    uint256 priceInAsset = getPriceFromCurvePool(pool, stor.index, stor.needInd

    uint256 assetPrice = priceRouter.getPriceInUSD(coins0);
    price = assetPrice.mulDivDown(priceInAsset, 10 ** coins0.decimals());
}
```

Reference: CurveEmaExtension.sol#L69-L78

```
function getPriceFromCurvePool(CurvePool pool, uint8 index, bool needIndex) pub
    return needIndex ? pool.price_oracle(index) : pool.price_oracle();
}
```

Reference: CurveEmaExtension.sol#L93-L95

However, the price returned by a curve pool's `price_oracle()` function is not using the base asset's decimals, since pools using USDT or USDC as their base asset return a price with 18 decimals, when these tokens both have 6 decimals. This can lead to incorrect price values for pools with a base asset that doesn't use 18 decimals.

Remediations to Consider

It seems like all prices returned from a curve pool's `price_oracle()` function uses 18 decimals, so instead of using the base asset's decimals, 18 could be used instead. Alternatively the price's decimals could be set as a value in the `extensionStorage` struct for each asset.

~~M-2~~ Share Price Oracles may not be properly initialized

TOPIC	STATUS	IMPACT	LIKELIHOOD
Error Recovery	Fixed	Medium	Low

The `ERC4626SharePriceOracle.sol`'s `initialize()` function a Chainlink `automationForwarder` is setup by registering a upkeep address for the contract by calling `registerUpkeep()` on Chainlink's registrar. It is assumed, however, that an upkeep will be registered when that call is made, but there are cases where registration is not accepted immediately and is set to pending, awaiting approval.

```
uint256 upkeepId;
if (_shouldAutoApprove(s_triggerRegistrations[params.triggerType], sender)) {
  s_triggerRegistrations[params.triggerType].approvedCount++;
  upkeepId = _approve(params, hash);
} else {
  uint96 newBalance = s_pendingRequests[hash].balance + params.amount;
  s_pendingRequests[hash] = PendingRequest({admin: params.adminAddress, balance
}
```

Reference: AutomationRegistrar2_1.sol#L419-L426

When a registration is set to pending, it will return a value of zero for its `upkeepId` and when the forwarder is queried form the registry with that id, the zero address is returned. This means that if the registrar's upkeep limit has been reached or if automatic upkeep approval is shut off, then share price oracles can not properly set a `automationForwarder`, as calls to the registry's `registerUpkeep()` will be set to pending, ending up with `automationForwarder` being set to the zero address, and no way of setting it to the proper address once the registration is approved.

Currently the upkeep auto approve limit on mainnet is 500, which can be increased or decreased, but if reached then all new registrations will have to await approval, preventing proper initialization of share price oracles.

Remediations to Consider

Add a function that can only be called if `initialize()` was called, but the `automationForwarder` address is still not set, and have this function verify a passed in upkeep id to ensure the id corresponds to the upkeep that was registered, then query the `automationForwarder` from the registry using that upkeep id. This will allow share price oracles to be setup if their upkeep isn't immediately approved. Also consider emitting an event if the `automationForwarder` isn't set when initialized.

~~L-1~~ Identifiers not updated for Frax Lend adaptors

TOPIC	STATUS	IMPACT	LIKELIHOOD
Protocol Design	Fixed	Low	High

The code was altered for the `DebtFTokenAdaptor`, `DebtFTokenAdaptorV1`, `CollateralFTokenAdaptor` and the `CollateralFTokenAdaptorV1` contracts. In order for these updated adaptors to be added to the registry they need to return a unique value from their `identifier()` function to distinguish them from the current Frax Lend adaptors.

```
/**
 * @notice Trust an adaptor to be used by cellars
 * @param adaptor address of the adaptor to trust
 */
function trustAdaptor(address adaptor) external onlyOwner {
    if (isAdaptorTrusted[adaptor]) revert Registry__AdaptorAlreadyTrusted(adapt
    bytes32 identifier = BaseAdaptor(adaptor).identifier();
    if (isIdentifierUsed[identifier]) revert Registry__IdentifierNotUnique();
    isAdaptorTrusted[adaptor] = true;
    isIdentifierUsed[identifier] = true;
}
```

Reference: Registry.sol#L309-L315

However, the `identifer()` function has not been updated along with these changes.

Remediations to Consider

Update the version number in each of the updated Frax Lend adaptor's `identifier()` functions to allow them to be added to the registry.

~~L-2~~ Missing event when initializing share price oracle

TOPIC	STATUS	IMPACT	LIKELIHOOD
Events	Fixed	Low	Medium

In `ERC4626SharePriceOracle.sol`'s `initialize()` function, a chainlink upkeep address is registered, and the corresponding `automationForwarder` address is set. There is no event emitted for this call, and the generated `upkeepId` returned when registering an upkeep is not emitted, and may be useful to know the `upkeepId` that corresponds to the share price oracle's `automationForwarder`, as useful data in Chainlink's registrar and registry is mapped to an `upkeepId`.

Remediations to Consider

Add an event that emits the `upkeepId` and `automationForwarder` address, when `initialize()` is called.

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 Sommelier 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.