macro

Sommelier A-11

Security Audit

Aug 30, 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 August 22, 2023 to August 24, 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
Code Quality	3	-	-	3
Gas Optimization	1	-	-	1

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.

Source Code

The following source code was reviewed during the audit:

- **Repository:** cellar-contracts
- **Commit Hash (initial):** `9e17494c2acd906718310b3e99fa6c1e918beaf6`
- **Commit Hash (final):** `0fa2a8ee2842eb7253439a3ee1f1acf5922c8c18`

Specifically, we audited the following contracts within this repository for the initial part of the audit, additionally the UniswapV3Adaptor was audited to ensure it would work with SushiswapV3 contracts.

Contract	SHA256
src/AxelarProxy.sol	b47e1da0a00263db9b2898713d61adfc254acc6e2ca988752ccaa78fce723fc6
src/modules/adaptors/Frax/CollateralFTokenAdaptorV1.sol	549e11be9b6c55073d92fb1f294ca50d483f52af983fd184fb339f5428e21019
src/modules/adaptors/Frax/CollateralFTokenAdaptorV2.sol	18de0265ca2067e6f1512f77c93bc0fec7784fab530ba42a433b8cb6ecedc63f
src/modules/adaptors/Frax/DebtFTokenAdaptorV1.sol	84342fc4dcd5fb9becc98d76573baaa267cce104755b597024d6873bfc4c3ee6
src/modules/adaptors/Frax/DebtFTokenAdaptorV2.sol	4b2273816b8c2a5816e9ebd0a61ead1982114fa6317429cb185705e5d977abf0
src/modules/adaptors/Frax/FTokenAdaptor.sol	b949ce28991511c63c095346858566b67986085f9caa56a3a0e4196fd4f2bbc5
src/modules/adaptors/Frax/FTokenAdaptorV1.sol	c3a2638038d5526d13882288c69a0e56ddc4292026348ab952d11b9d7f09895f

Contract	SHA256
src/modules/adaptors/Frax/FraxlendHealthFactorLogic.sol	2a6476ab594bea4c2438dcef069e53e158b203d7c06b5f2244f6c438518edf55
src/modules/adaptors/Uniswap/UniswapV3Adaptor.sol	02f12375ad25274a4c827fa63a74c44b80881e06eea99e0efc2eb2fb8a5e162b
src/modules/adaptors/Uniswap/UniswapV3PositionTracker.sol	bf1f09a14ad795bce2986bbb760daee45e4d2750384e03c4d50b6d74b9e4f5de

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~~ Wrong exchange rate used for health factor calculations can bring positions close to liquidation

TOPIC	STATUS	IMPACT	LIKELIHOOD
Input Parameters	Fixed	Medium	Medium

In both `CollateralFTokenAdaptorV2.sol` and `DebtFTokenAdaptorV2.sol` the exchange rate used to calculate the health factor of the position is the `lowExchangeRate`.

```
function _updateExchangeRate(IFToken _fraxlendPair) internal virtual returns (u
    (, exchangeRate, ) = _fraxlendPair.updateExchangeRate();
}
```

Reference: CollateralFTokenAdaptorV2.sol#L235-L237 and DebtFTokenAdaptorV2.sol#L284-L286

```
function updateExchangeRate()
    external
    nonReentrant
    returns (bool _isBorrowAllowed, uint256 _lowExchangeRate, uint256 _highExch
{
    return _updateExchangeRate();
}
```

Reference: FraxLendPairCore.sol#L480-L486

However in Frax Lend, when calculations are done to check if a user is solvent, which is what the code for `_getHealthFactor()` is based on, it uses the `highExchangeRate`.

```
modifier isSolvent(address _borrower) {
    _;
    ExchangeRateInfo memory _exchangeRateInfo = exchangeRateInfo;

    if (!_isSolvent(_borrower, exchangeRateInfo.highExchangeRate)) {
        revert Insolvent(
            totalBorrow.toAmount(userBorrowShares[_borrower], true),
            userCollateralBalance[_borrower],
            exchangeRateInfo.highExchangeRate
        );
    }
}
```

Reference: FraxLendPairCore.sol#L236-L247

Using a the lower exchange rate reduces the Loan to Value (LTV) calculated and thus increases the health factor calculated. Since the minimum health factor is set to ensure the position remains a set percent above the liquidation threshold, this may cause the position to be allow to get closer to insolvency than desired if there is a sufficient deviation between the high and low exchange rate.

Remediations to Consider

Use the `highExchangeRate` when calling `_getHealthFactor()` to use the same value that is used to calculate if a position is solvent, and prevent a cellars position from getting too close to insolvency.

~~M-2~~ Strategist can execute arbitrary code, potentially extracting value from the cellar

TOPIC	STATUS	IMPACT	LIKELIHOOD
Input Validation	Fixed	Medium	Low

In `CollateralFTokenAdaptorV2.sol`'s `removeCollateral()` function, there is no check to see of the passed in `_fraxlendPair` is a valid position used by the cellar.

```
function removeCollateral(uint256 _collateralAmount, IFToken _fraxlendPair) pub
  // remove collateral
  _removeCollateral(_collateralAmount, _fraxlendPair);
  uint256 _exchangeRate = _updateExchangeRate(_fraxlendPair); // need to calcu
  // Check if borrower is insolvent (AKA they have bad LTV), revert if they are
  if (minimumHealthFactor > (_getHealthFactor(_fraxlendPair, _exchangeRate))) {
      revert CollateralFTokenAdaptor__HealthFactorTooLow(address(_fraxlendPair)
  }
}
```

Reference: CollateralFTokenAdaptorV2.sol#L151-L159

Since calls are directly made to this address, it could be any contract that isn't verified. This can allow a contract to execute arbitrary code that is unverified. This contract call could interact with protocols the cellar has positions in and potentially effect the outcome of other calls executed within the same cellar `callOnAdaptor` call, potentially extracting value from the cellar.

This issue is also present in `FTokenAdaptor.sol` 's `withdraw()` function.

Remediations to Consider

Call `_validateFToken()` on the passed in `_fraxlendPair` in `CollateralFTokenAdaptorV2.sol` 's `removeCollateral()` function and `FTokenAdaptor.sol` 's `withdraw()` function, to ensure the pair is a valid contract that the cellar has a position in.

~~L-1~~ Using uint256.max as a `nonce` can prevent future strategist and governance calls to the cellar

TOPIC	STATUS	IMPACT	LIKELIHOOD
Input Ranges	Fixed	High	Low

In `AxelarProxy.sol`, each call to execute requires that the nonce provided is greater than the last nonce used to call a target cellar. These nonces are incremented on the protocol side on each transaction. However, since there is limit on the nonce provided, if it is set for a cellar as the max value of `uint256`, then no further calls can be made by the proxy to that cellar, since the nonce cannot exceed it. This situation is unlikely to occur as the nonces should be properly handled by the protocol, but there should be a bound placed to ensure the `nonce` provided is reasonable and will not reach the max `unit256` value.

Remediations to Consider

Add a reasonably large bound on the `nonce` provided from the previously set `nonce`, something that could not be exceeded by regular means, to ensure calls can always be made to it cellars via the proxy.

~~L-2~~ Unnecessary use of `minimumNonce`

TOPIC	STATUS	IMPACT	LIKELIHOOD
Protocol Design	Fixed	Low	Low

In `AxelarProxy.sol`, a `minimumNonce` is setup in the constructor, and is used to ensure that nonces below that value cannot be used in `_execute()`.

```
if (nonce < minimumNonce) revert AxelarProxy__MinimumNonceUnmet();
```

Reference: AxelarProxy.sol#L74

The intent behind setting the `minimumNonce` is to invalidate prior accepted function calls in the case a new `AxelarProxy` is deployed, in case of an update or there is a new `AxelarGateway` contract, with the intention of the `minimumNonce` being set above the current protocols nonce as a buffer to

However, each call that is accepted on the `AxelarGateway` contract is specific to a target `destinationAddress`, which when the `AxelarProxy` validates the call when executing, it is only valid if it is the set destination address, since `msg.sender` is used to generate the key when `validateContractCall()` is called on the `AxelarGateway`.

```
function validateContractCall(
    bytes32 commandId,
    string calldata sourceChain,
    string calldata sourceAddress,
    bytes32 payloadHash
) external override returns (bool valid) {
    bytes32 key = _getIsContractCallApprovedKey(commandId, sourceChain, sourceA
    valid = getBool(key);
    if (valid) _setBool(key, false);
}
```

Reference: AxelarGateway.sol#L229-L238

This means that old validated contract calls, are not valid for the new AxelarProxy, meaning there is not need to set a `minimumNonce`.

Since the `nonce` used when making these calls is tied to the number of proposals made on the sommelier side, setting the minimum nonce to a large value could make it difficult to execute calls on the `AxelarProxy` as nonce would have to be incremented above the minimum nonce set before calls can execute.

Remediations to Consider

Remove the use of `minimumNonce` as it is not required and could cause issues if improperly set.

~~Q-1~~ Inaccurate comment

TOPIC	STATUS	QUALITY IMPACT

Code Quality	Fixed	Low

In `FraxlendHealthFactorLogic.sol`'s `_getHealthFactor()` there is a comment that mentions fraxlend's "getHealthFactor() function", however this may be referring to fraxlend's `isSolvent()` function instead.

```
// need interest-adjusted and conservative amount (round-up) similar to `_getHe
```

Reference: FraxlendHealthFactorLogic.sol#L27

Remediations to Consider

Adjust the comment to be more accurate.

~~Q-2~~ `_fraxlendPairAsset` is defined but not used

TOPIC	STATUS	QUALITY IMPACT
Code Quality	Fixed	Low

In `CollateralFTokenAdaptorV2.sol` the function `_fraxlendPairAsset()` is defined, but is never called.

Remediations to Consider

Remove `_fraxlendPairAsset()` since it is not used.

~~Q-3~~ Rename Frax V2 adaptors

TOPIC	STATUS	QUALITY IMPACT
Code Quality	Fixed	Low

To follow the pattern of `FTokenAdaptor.sol` instead of differentiating adapters as `V2` in the cases of `DebtFTokenAdaptorV2.sol` and `CollateralFTokenAdaptorV2.sol`.

Remediations to Consider

Rename `DebtFTokenAdaptorV2.sol` to `DebtFTokenAdaptor.sol` and `CollateralFTokenAdaptorV2.sol` to `CollateralFTokenAdaptor.sol` to maintain the naming pattern set by `FTokenAdaptor.sol`.

~~G-1~~ Frax Lend exchange rates do not need to be updated

TOPIC	STATUS	GAS SAVINGS
Gas Optimization	Fixed	Low

In the Frax Lend adaptors, `_updateExchangeRate()` is called in the collateral adaptors after `removeCollateral()` is called, and is called by the debt adaptors after `_borrowAsset()` is called. In both versions of Frax Lend when either collateral is removed by calling `removeCollateral()` or an asset is borrowed by calling `borrowAsset()`, the exchange rate is updated. Since the exchange rate can only be updated once per block, calling `updateExchangeRate()` will have no effect. Instead, in order to get the updated exchange rate, a call to `exchangeRateInfo()` will return an `ExchangeRateInfo` struct, which does vary from versions V2 and V1, and the updated exchange rate can pulled from it.

Remediations to Consider

Call the view function `exchangeRateInfo()` on the `FraxLendPair` instead of `updateExchangeRate()` to save a bit of gas.

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.