PUBLIC

Code Assessment

of the Extensions III Smart Contracts

March 14, 2022

Produced for

by

Contents

1 Executive Summary

Dear Mona and Sean,

Thank you for trusting us to help Avantgarde Finance with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Extensions III according to Scope to support you in forming an opinion on their security risks.

The present report covers the implementation of the Fuse lending adapter, the Aave borrowing external position, the upgrade in the UniswapV3 external position where all liquidity NFTs are handled by the same external position and the USD to ETH price oracle. All in all, only one medium importance issue was uncovered where the remaining amount after a loan repayment remains locked in the external position. Furthermore, a few more minor design issues are reported. All the issues have been corrected.

The most critical subjects covered in our audit are functional correctness and access control. Security regarding all the aforementioned subjects is high.

The general subjects covered are upgradability, unit testing and gas efficiency. Security regarding all the aforementioned subjects is high. The specification provided was comprehensive.

In summary, we find that the aforementioned modules to be added to the system of a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	1
• Code Corrected	1
Low-Severity Findings	3
• Code Corrected	3

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Extensions III repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	28 February 2022	1dca776e53254289b5b40ba57173549bce47e3e2	Initial Version
2	14 March 2022	91f3b146ae6667981a14c3918efb2917b7b6435f	Second Version

For the solidity smart contracts, the compiler version `0.6.12` was chosen, for the smart contracts related to UniswapV3 LP the compiler version `0.7.6` was chosen. More specifically, the files in scope are the following:

- contracts/persistent/external-positions/uniswap-v3-liquidity/UniswapV3LiquidityPositionLibBase1.sol
- contracts/release/extensions/external-position-manager/external-positions/uniswap-v3-liquidity:
  - UniswapV3LiquidityPositionDataDecoder.sol
  - UniswapV3LiquidityPositionLib.sol
  - UniswapV3LiquidityPositionParser.sol
- contracts/release/infrastructure/price-feeds/primitives/UsdEthSimulatedAggregator.sol
- contracts/persistent/external-positions/aave-debt/AaveDebtPositionLibBase1.sol
- contracts/release/extensions/external-position-manager/external-positions/aave-debt/:
  - AaveDebtPositionDataDecoder.sol
  - AaveDebtPositionLib.sol
  - AaveDebtPositionParser.sol
- contracts/release/extensions/integration-manager/integrations/adapters/:
  - CompoundAdapter.sol
  - FuseAdapter.sol
- contracts/release/extensions/integration-manager/integrations/utils/bases/:
  - CompoundAdapterBase.sol
- contracts/release/infrastructure/price-feeds/derivatives/feeds/:
  - FusePriceFeed.sol
  - utils/CompoundPriceFeedBase.sol

, as well as their respective interfaces.

2.1.1 Excluded from scope

Files not listed above and the external systems are not in scope.

2.2 System Overview

This system overview describes the changed and new functionality reviewed as part of this report, see the Assessment Overview. For the main system description and trust model, please refer to the main audit of the Sulu release. For the external UniswapV3 positions please refer to the corresponding UniswapV3 external position extension audit.

2.2.1 External UniswapV3 position changes

Before, fund managers were required to deploy an external position proxy per token pair. Now, all UniswapV3 positions are handled through the same external position proxy to reduce gas cost. The following changes were made to the contracts:

- `token0` and `token1` variables were replaced by a mapping that map NFT ids (position ids) to the tokens.
- Initialization code of the proxy is removed since the tokens are not fixed. Hence, the token approvals to Uniswap's NonfungiblePositionManager are given lazily on minting a position.
- The `getManagedAssets()` function now aggregates the underlying tokens to correctly account for introduced functionality.
- Of course the correct functionality of the external position depends on the existence of the respective price feeds. Should a price feed for a token be removed while the external position still holds the token, its functionality might break.

2.2.2 External Aave variable debt positions

Avantgarde Finance implements an external position for borrowing funds for sulu using Aave-v2 Protocol. Through Aave, users can use aTokens (tokens issued by the Aave protocol) as collateral to borrow funds. Fund managers can perform the following actions using this external position:

- Add Collateral: This allows managers to transfer an amount of an asset their vault holds to the external position and then let Aave know that this asset can be used as collateral. Finally, it updates the collateral assets array.
- Borrow Amount: Allows managers to borrow an amount of an asset belonging to the Enzyme asset universe using the collateral they have assigned to the external position. The borrowed amount is then transferred to the vault. The array of the borrowed assets is updated.
- Repay Amount: Allows managers to repay part of their debt. An amount is transferred from the vault to the external position and then transferred to Aave. If an asset is fully repaid, then it is removed from the borrowed assets array.
- Remove Collateral: Withdraws an amount of aTokens from the external position back to the vault. It is important to note that the transfer will fail in case it leaves the borrowed position undercollateralized.
- Claim Rewards: Aave can incentivize aTokens and debt tokens. Such tokens collect rewards which can be claimed through Aave’s incentive controller. The rewards are claimed directly for the vault.
- The same concern mentioned for UniswapV3 regarding the available price feeds holds for Aave. Removing price feeds might prohibit the external position from correctly functioning.

2.2.3 Fuse lending integration

Avantgarde Finance implements an adapter for Rari Fuse which is a Compound fork. Hence, the existing Compound adapter code has been modified and, additionally, reward claiming functionality has been added.

- The content of the Compound adapter has been moved to the base adapter `CompoundAdapterBase`, an abstract contract not implementing functionality for claiming rewards.
- The Compound adapter now extends `CompoundAdapterBase` and implements `claimRewards()` which calls `claimComp()` on the Compound comptroller to receive rewards.
- Fuse adapter was introduced and, similarly, extends `CompoundAdapterBase` to implement `claimRewards()` which call `claimRewards()` on the Fuse comptroller and then iterates over all Fuse's reward distributors.
- The Compound price feed has been copied into `CompoundPriceFeedBase` without semantic changes. Fuse price feed extends the base contract without implementing any semantic changes.
- Note that the compound price feed remains unchanged.

The functionality of the adapters implements lending, redeeming and claiming rewards.

2.2.4 Simulated USD/ETH Chainlink aggregator

So far the USDC price feed has been used to evaluate USD prices. However, to estimate USD prices more accurately, a price feed for USD/ETH is introduced. The following methods are implemented:

- `decimals()`: returns 18 - the number of decimals used for the return value of the oracle.
- `latestRoundData()`: Returns the round id, the price, the timestamps of when the Chainlink round started and ended and the round id in which the answer was computed. Note, that Chainlink offers a ETH/USD price feed and the price returned by the wrapper will be the inverted value of Chainlink's price.

2.2.5 Trust Model

Please refer to the main audit report for a general trust model of Sulu.

Rari Fuse: Fuse is trusted and expected to work as Compound.

Aave V2: Aave is trusted and is expected to work as documented.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	1

- Locked Assets After Repay Code Corrected

Low-Severity Findings	3

- Rebasing aToken Balance Code Corrected
- Redundant Call Code Corrected
- Sanity Check Missing Code Corrected

6.1 Locked Assets After Repay

Design Medium Version 1 Code Corrected

A manager could try to repay an amount bigger than what they owe to Aave. In such cases, the leftover amount will not be transferred back to the vault. Consider the following case:

1. A manager owes 100 DAI
2. The manager tries to repay 110 DAI
3. 110 DAI will be transferred from the vault to the external position
4. The call to Aave will only consume 100 DAI. The remaining amount will remain in the external position

---

Code corrected:

After the debt owed to Aave is repaid, the remaining balance of the repayment token is sent back to the vault proxy. Since the repayment token is an underlying and not an aToken, the transfer will not affect the health factor of the positions.

6.2 Rebasing aToken Balance

Design Low Version 1 Code Corrected

ATokens are rebasing tokens. This means that the balance an account holds changes in time and, thus, between the time a transaction is submitted and mined. In the current implementation, there is no way to remove the full amount of the collateral by querying the balance the external position holds during the execution of the transaction. This could result in dust remaining in the external position.

Code corrected:

By specifying the maximum integer as the amount, the full aToken balance will be withdrawn:

```
uint256 collateralBalance = ERC20(aTokens[i]).balanceOf(address(this));

if (amounts[i] == type(uint256).max) {
    amounts[i] = collateralBalance;
}

// If the full collateral of an asset is removed, it can be removed from collateral assets
if (amounts[i] == collateralBalance) {
    collateralAssets.removeStorageItem(aTokens[i]);
    emit CollateralAssetRemoved(aTokens[i]);
}
```

6.3 Redundant Call

Design | Low | Version 1 | Code Corrected

`__addCollateralAsset` call the lending pool function `setUseReserveAsCollateral` which enables an asset to be used as a collateral. However, the implementation of regular transfers will automatically use the underlying of the transferred aToken as collateral if a zero-balance is increased (see AToken code and lending pool's finalizeTransfer). Hence, the call may be redundant.

Code corrected:

The call has been removed.

6.4 Sanity Check Missing

Correctness | Low | Version 1 | Code Corrected

An added collateral token is never sanitized. Assume a malicious manager to create an evil token which is compatible with the interface of the AToken, i.e. it exposes a method `UNDERLYING_ASSET_ADDRESS()` which returns a token used by Aave. Since this token is never sanitized, it could be added as collateral since the call

```
(lendingPoolAddress).setUserUseReserveAsCollateral(AaveAToken(aTokens[i]).UNDERLYING_ASSET_ADDRESS(), true);
```

will succeed. Adding such a token, however, could block the function `ControllerLib.calcGaV()` which calculates the external position value. During the calculation, the managed assets are queried with `getManagedAssets` in order to be priced but no price feed for the evil token exists.

Code corrected:

The AaveDebtPositionParser will now validate that the token added as collateral is a whitelisted token. Ultimately supported non-aTokens could be deposited. However, that does not block execution nor could it lock tokens.

7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

7.1 Aave Paused

Note Version 1

The lending pool of Aave could be paused and, hence, actions on the vault will not be possible to execute. Ultimately, positions could be not modifiable, and funds could be stuck.

7.2 Sandwiching Transactions for Liquidation

Note Version 1

It is known that the behavior of the managers is monitored. However, we would like to point out that there are sequences of actions from which the manager can benefit. In particular, a malicious fund manager, who sees a price drop in the Aave oracle of a collateral asset, could create a malicious sequence of transactions through MEV capabilities to borrow with user funds while liquidating the position immediately. Consider the following sequence of transactions:

1. Move aDai to the Aave external position proxy and borrow WETH such that the health factor is 1.
2. The sandwiched oracle price changes: Dai price drops compared to WETH.
3. The fund manager liquidates the position and profits.