

Produced for



by



# Contents

<b>1</b>	<b>Executive Summary</b>	<b>3</b>
<b>2</b>	<b>Assessment Overview</b>	<b>5</b>
<b>3</b>	<b>Limitations and use of report</b>	<b>10</b>
<b>4</b>	<b>Terminology</b>	<b>11</b>
<b>5</b>	<b>Findings</b>	<b>12</b>
<b>6</b>	<b>Resolved Findings</b>	<b>13</b>
<b>7</b>	<b>Notes</b>	<b>15</b>

# 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 Sulu Extensions VIII according to [Scope](#) to support you in forming an opinion on their security risks.

Avantgarde Finance implements new adapters for Aave V3 and Compound V3 and refactors the codebase of the Aave V2 adapter so that code can be reused for the Aave V3 adapter. Additionally, Avantgarde Finance introduces so-called list owner contracts, used for validation in the aforementioned adapters, that can add validated items to a list. Further, Avantgarde Finance implements an upgrade for the Maple external position to allow migration to V2.

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

The general subjects covered are code complexity, upgradeability, and documentation. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides 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.

-Severity Findings	0
-Severity Findings	1
•	1
-Severity Findings	0
-Severity Findings	2
•	1
•	1



## 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 Sulu Extensions VIII 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	21 November 2022	10e16f2b6f0e7b170ae81027dd446c1919920d8f	Initial Version
2	30 November 2022	320200f16e952e8b4163bbdc36620373fc7aa689	After Intermediate Report
3	07 December 2022	97b2b6eec6f48ad896df305d080faaff3d8b710e	Final Version

For the solidity smart contracts, the compiler version 0.6.12 was chosen.

The following files and directories were in scope

#### Maple External Position

```
contracts/persistent/external-positions/maple-liquidity/  
contracts/release/extensions/external-position-manager/external-positions/maple-liquidity/  
contracts/release/interfaces/IMapleV1MplRewards.sol  
contracts/release/interfaces/IMapleV1MplRewardsFactory.sol  
contracts/release/interfaces/IMapleV1Pool.sol  
contracts/release/interfaces/IMapleV2Pool.sol  
contracts/release/interfaces/IMapleV2PoolManager.sol  
contracts/release/interfaces/IMapleV2ProxyFactory.sol  
contracts/release/interfaces/IMapleV2WithdrawalManager.sol  
contracts/release/interfaces/IERC4626.sol
```

#### Address List Registry Owners

```
contracts/persistent/address-list-registry/address-list-owners/
```

#### Aave Adapters and Debt Position

```
contracts/persistent/external-positions/aave-v2-debt/AaveDebtPositionLibBase1.sol  
contracts/release/extensions/external-position-manager/external-positions/aave-v2-debt/  
contracts/release/extensions/integration-manager/integrations/adapters/AaveV2Adapter.sol  
contracts/release/extensions/integration-manager/integrations/adapters/AaveV3Adapter.sol  
contracts/release/extensions/integration-manager/integrations/utills/actions/AaveActionsMixin.sol  
contracts/release/extensions/integration-manager/integrations/utills/actions/AaveV2ActionsMixin.sol  
contracts/release/extensions/integration-manager/integrations/utills/actions/AaveV3ActionsMixin.sol  
contracts/release/extensions/integration-manager/integrations/utills/bases/AaveAdapterBase.sol  
contracts/release/infrastructure/price-feeds/derivatives/feeds/AavePriceFeed.sol  
contracts/release/interfaces/IAaveAToken.sol  
contracts/release/interfaces/IAaveLendingPool.sol
```

```
contracts/release/interfaces/IAaveV2IncentivesController.sol
contracts/release/interfaces/IAaveV2LendingPool.sol
contracts/release/interfaces/IAaveV2LendingPoolAddressProvider.sol
contracts/release/interfaces/IAaveV2ProtocolDataProvider.sol
contracts/release/interfaces/IAaveV3Pool.sol
contracts/release/interfaces/IAaveV3PoolAddressProvider.sol
```

## Compound V3 Adapter

```
contracts/release/extensions/integration-manager/integrations/adapters/CompoundV3Adapter.sol
contracts/release/extensions/integration-manager/integrations/utils/actions/CompoundV3ActionsMixin.sol
contracts/release/interfaces/ICompoundV3Comet.sol
contracts/release/interfaces/ICompoundV3CometRewards.sol
contracts/release/interfaces/ICompoundV3Configurator.sol
```

## WstETH Price Feed

```
contracts/release/infrastructure/price-feeds/derivatives/feeds/WstethPriceFeed.sol
contracts/release/interfaces/ILidoSteth.sol
```

### 2.1.1 Excluded from scope

All files not mentioned above are not in scope. Maple, Aave, Compound, and Lido are not in scope and are expected to work correctly as documented.

## 2.2 System Overview

This system overview describes the initially received version ( ) of the contracts as defined in the [Assessment Overview](#).

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Avantgarde Finance implements integrations with various external systems. Some integrations were newly added, others were refactored.

### 2.2.1 External Position: Maple Finance v2

Maple will migrate to V2 of their contract. We describe Avantgarde Finance's external position for V2 during migration and after migration.

#### 2.2.1.1 Considerations for the Migration to V2

Maple Finance is upgrading its protocol, which means migration from V1 to V2 is necessary. In this section, we outline the relevant technical details to facilitate the migration.

1. Before Maple starts their migration, the Enzyme council will upgrade the library for the external position. Additionally, for the coordination of the Migration on Avantgarde Finance's side, the contract `MapleV1ToV2PoolMapper` will be deployed.
2. While `MapleV1ToV2PoolMapper` allows for snapshots to be taken, the function `snapshotPoolTokenV1BalanceValues()` should be called (callable by anyone and also multiple times) on every external position. This stores the total value of the position in terms of the liquidity asset such that the position's value can be robustly estimated during migration. Note that this estimates the value in the pool in the same way that the `getManagedAssets()` for V1 did.

Also note that `MapleV1ToV2PoolMapper` offers a batched function `snapshotExternalPositions()` that calls `snapshotPoolTokenV1BalanceValues()` on multiple external positions.

3. Council calls `MapleV1ToV2PoolMapper.freezeSnapshots()` to disallow creating new snapshots.
4. Maple runs their migration, which airdrops Maple V2 LP tokens.
5. The council calls `MapleV1ToV2PoolMapper.mapPools()` to map V1 to V2 pools.
6. Once Maple has cross-checked the mapping and the council has validated that each external position received a valid amount of V2 LPs, the council calls `MapleV1ToV2PoolMapper.allowMigration()` to allow migrations to happen.
7. Anyone can migrate any V1 position by using `migrateV1ToV2Pools()` (only possible after 6.). Note that there is a batched function `MapleV1ToV2PoolMapper.migrateExternalPositions()`. This initializes the used V2 pools by mapping the used V1 pools to V2 pools using `MapleV1ToV2PoolMapper.getPoolTokenV2FromPoolTokenV1()` and untracks V1 pools and deletes their snapshots.

To summarize, all external positions have some shared state (`snapshotsAllowed`, `migrationDisallowed`) and some individual state defining whether it has completed migration. The values will change according to the following order

1. (`true`, `true`) and position migration not completed - initialization. `getManagedAssets()` will account for V1 positions by taking snapshots and returning them.
2. (`false`, `true`) and position migration not completed - `MapleV1ToV2PoolMapper.freezeSnapshots()`. `getManagedAssets()` will account for V1 positions according to the snapshots taken.
3. (`false`, `false`) and position migration not completed - `MapleV1ToV2PoolMapper.allowMigration()`. `getManagedAssets()` will automatically migrate the position and will not account for V1 tokens anymore.
4. (`false`, `false`) and position migration completed - `ExternalPosition.migrateV1ToV2Pools()`. `getManagedAssets()` will not account for V1 tokens anymore.

### 2.2.1.2 External Position

This section focuses on the integration with Maple Finance v2. Their special behavior during migration is described in the section above. Note that Maple V2 can be interacted with even during the migration. However, `RequestRedeemV2` is blocked since redemption could allow untracked airdropped V2 tokens to be redeemed.

The following actions are available for V2:

- `LendV2`: Deposits the Maple pool's underlying asset (`IMapleV2Pool.asset()`) with `IMapleV2Pool.deposit()` into the Maple pool and starts tracking the pool if it was untracked. The parser validates the pool against the pool proxy factory.
- `RequestRedeemV2`: Redemptions must first be requested. Shares will be escrowed (not necessarily all shares specified). The parser validates the pool against the pool proxy factory.
- `CancelRedeemV2`: Cancels redemption requests. Hence, funds are moved from Maple's withdrawal manager to the position. The parser validates the pool against the pool proxy factory.
- `RedeemV2`: Redeems locked shares and moves funds to the vault proxy. The parser validates the pool against the pool proxy factory.

Note that the pool validation is done with the pool factory's `isInstance()` function.

The action `ClaimRewards` was renamed to `ClaimRewardsV1` but has functionally not changed. This action allows the collection of legacy rewards on V1.

`getManagedAssets()` iterates over the used pools and computes the value of the held and locked shares in terms of the underlying asset. Note that the managed value may include the V1 value as described in the previous section. `getDebtAssets()` will return empty arrays since no debt is created.

## 2.2.2 Add-Only List Contract Base Infrastructure

Two new abstract contracts are introduced to create address list owners with validation checks and to interact with them.

- `AddOnlyAddressListOwnerBase`: On construction, the add-only list owner contracts create an add-only list in the address list registry and attest ownership for the list. Note that items can be added by anyone by default with the function `addValidatedItemsToList()` which validates the items with the abstract function `__validateItems()` that derived contracts must implement. Note that `addValidatedItemsToList()` by default does not have any access control. This can be added in derived contracts if necessary.
- `AddOnlyAddressListOwnerConsumerMixin`: Contracts inheriting from this mix-in will be provided with the internal function `__validateAndAddListItemIfUnregistered()` which ensures that after its successful execution the item is in the list owned by the owner contract specified on construction.

## 2.2.3 Aave V2 and V3

Avantgarde Finance introduces an integration adapter for Aave V3. Note that some refactoring was performed for Aave V2-related files performed due to renamings of files. Both adapters share the same actions:

- `lend()`: Validates (or adds the lent token address to the add-only list) and lends out tokens.
- `redeem()`: Validates (or adds the lent token address to the add-only list) and redeems tokens.

For the Aave V2 adapter, the most notable change (compared to the previous version) is that the validation of aTokens is performed with a list and not, as before, with a price feed. For the Aave V3 adapter, the tokens are lent with `Pool.supply()` and are redeemed with `Pool.withdraw()`.

Both the V2 and the V3 add-only list owner contract, check aToken addresses against the respective pool's reserve data when adding new items to the list.

Note that Aave V2 and V3 LPs will be registered to use their underlying's price feeds since the exchange rate of aToken to underlying is 1:1.

## 2.2.4 Compound v3

Compound v3 is a lending protocol that enables borrowing a base asset against different collateral assets. Avantgarde Finance implements an adapter that allows depositing the base asset into Compound v3.

The adapter supports three actions:

- `lend()`: Validates the cToken (Comet) and spends the Comet's base asset to mint LPs to the vault proxy.
- `redeem()`: Validates the cToken (Comet) and burns LPs to receive the Comet's base asset.
- `claimRewards()`: Calls `claim()` on the rewards contract to claim rewards for the vault proxy.



## 2.2.5 Lido wstETH Price Feed

Avantgarde Finance introduces a new price feed that computes wstETH's value in stETH. wstETH reflects the underlying shares system of stETH and, hence, can be priced through the stETH contract's `getPooledEthByShares()` function.

## 2.2.6 Roles and Trust Model

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

Governance is fully trusted and expected to not only behave honestly but also to fully understand the systems they are interacting with which includes choosing appropriate parameters.

Fund owners and asset managers are generally fully trusted for a fund. However, their powers can be limited through the fund's settings.

All external systems are expected to be non-malicious and work correctly as documented. For Maple, we assume that the migration is handled as specified, `convertToExitAssets()` is not manipulatable, and that not redeeming requests has no impact on the value of the position. For Aave and Compound, we assume that rounding errors will be minor. For Lido, we expect the `getPooledEthByShares()` to be not manipulatable.

In general, we assume Enzyme only interacts with normal ERC-20 tokens that do not have multiple entry points or callbacks.

## 2.2.7 Changes in Version 2 of the code base

- The actions for Maple V2 can only be executed after migration.
- If a Maple position has not been migrated, the actions for Maple V2 try to migrate it (if possible). Earlier, these reverted and required a call to `migratePoolsV1ToV2()` beforehand.

## 2.2.8 Changes in Version 3 of the code base

Avantgarde Finance notified us that pools will not be deployed through the `MapleProxyFactory`. Pools will be deployed through a `PoolDeployer` contract with function `deployPool()` which deploys pools during the initialization of the `PoolManager` contracts. Hence, the validity checks were updated to ensure that pools and their pool managers match and to validate that a pool's pool manager has been deployed by a valid factory.

Note that we assume that `deployPool` deploys a `PoolManager` contract so that it can be validated with `isInstance()`. Further, we assume that the pool manager factory is stored with key `"POOL_MANAGER"` in Maple's `Globals` contract and that a factory, once activated, will not be deactivated. In case a factory is deactivated, we expect that Maple clearly communicates such changes to Avantgarde Finance so that Avantgarde Finance has enough time to find a solution if that is a breaking change (validity checks fail for some position).

### 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			
Medium			
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:

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

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

-Severity Findings	0
-Severity Findings	0
-Severity Findings	0
-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.

-Severity Findings	0
--------------------	---

-Severity Findings	1
--------------------	---

- [Double Counting During Maple Migration](#)

-Severity Findings	0
--------------------	---

-Severity Findings	2
--------------------	---

- [Incorrect Comment](#)
- [Unused Import](#)

### 6.1 Double Counting During Maple Migration

During the migration of Maple positions, double counting of Maple LP tokens is possible as there are no restrictions enforced on `lend()`.

Consider the following scenario:

1. The position holds 10 v1 LP tokens.
2. The snapshot is taken and snapshots are frozen.
3. The airdrop of v2 LP tokens happens and the position receives 10 v2 LP tokens. Note that `getManagedAssets()` does not consider v2 LP tokens since the v2 pool is not tracked. Hence, the valuation is 10 v1 LP tokens.
4. The manager lends tokens to Maple v2 and creates 10 v2 LP tokens. Now, `getManagedAssets()` considers both v1 and v2 LP tokens since lending will start tracking the v2 pool. Hence, the valuation is 10 v1 LP tokens and 20 v2 LP tokens.

Thus, funds could be overvalued between airdrop and migration execution.

---

#### Code corrected:

Lending is now only allowed if the position has been migrated.

### 6.2 Incorrect Comment

On the `__validateAndAddListItemIfUnregistered` function of `AddOnlyAddressListOwnerConsumerMixin` there is the following comment:

```
/// @dev Helper to lookup an item's existence and then attempt to add it.  
/// AddOnlyAddressListOwnerBase.addToList() performs validation on the item.
```

The `addToList` function does not actually perform the validation. The `__validateItems` function does.

---

**Specification changed:**

The comment now specifies that the function `addValidatedItemsToList()` is used.

## 6.3 Unused Import

`MapleV1ToV2PoolMapper` imports `"@openzeppelin/contracts/token/ERC20/ERC20.sol"` which is unused.

---

**Code corrected:**

The import has been removed.

## 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 Lido Rebasing

Lido has epochs for rebasing. It could be possible to sandwich oracle updates with buying and selling shares.

### 7.2 Maple V2 Migrations Can Start Before Snapshots

`allowMigration()` is a governance function that should be called after all snapshots. Note that there is no sanity check that all snapshots have been made. Governance should not call this function too early.

Further, note that the function should only be called after snapshots have been disallowed with `freezeSnapshots()`.

### 7.3 Pool Address

Note that for the Aave v2 and Aave v3 adapters, the lending pool's address is stored and not queried from Aave's registry. Managers should be aware that the (lending) pool address used could be outdated.

---

Avantgarde Finance plans to upgrade the library contract in the case that the lending pool address changes.

### 7.4 Potential Maple V2 Rollback

Note that Maple V2 could rollback their migration process. Avantgarde Finance should be aware and quick to react.

### 7.5 Redeemable Amount

The action `RedeemV2` tries to redeem the input amount `poolTokenAmount`. However, note that Maple's `WithdrawalManager` contains the following code

```
require(requestedShares_ == lockedShares_, "WM:PE:INVALID_SHARES");
```

Hence, managers should be aware that RedeemV2 will only succeed if `poolTokenAmount` is equal to its locked shares.

---

Avantgarde Finance prefers this in case future implementations of Maple change this logic.

## 7.6 Reverts on Batching Maple Migration

The Maple mapper contract implements a batched function for migrating to v2. If a position has already been migrated, the batched function may revert.

---

Avantgarde Finance replied:

```
Pretty unlikely to occur and nicer to be able to easily preview a tx failure  
by not skipping reverting items.
```

## 7.7 Theoretical Out-Of-Gas During Maple V2 Migrations

`snapshotPoolTokenV1BalanceValues()` and `migratePoolsV1ToV2()` load all used pools from storage. Theoretically, these functions could result in an out-of-gas problem that cannot be resolved without a contract upgrade.

In contrast, if too many pools are added for `getManagedAssets()`, this can be resolved by a manager.