PUBLIC

Code Assessment

of the Sulu AaveV3 Flashloan Smart Contracts

October 3, 2024

Produced for

by

Contents

1 Executive Summary

Dear Enzyme Team,

Thank you for trusting us to help Enzyme Foundation with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Sulu AaveV3 Flashloan according to Scope to support you in forming an opinion on their security risks.

Enzyme Foundation implements a smart account for AaveV3 flash loans. Only flash loans with `interestRateMode` 0 can be performed, enforcing the flash loan to revert if not payed back directly.

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

The audit did not uncover any issues.

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.

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

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 AaveV3 Flashloan 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	29 September 2024	c5acff2b309b8940de604d0aecdfbaa12ce332a2	Initial Version

For the solidity smart contracts, the compiler version `0.8.19` was chosen.

2.1.1 Included in scope

• contracts/persistent/smart-accounts/aave-v3-flash-loan-asset-manager/AaveV3FlashLoanAssetManagerLib.sol

2.1.2 Excluded from scope

Any contracts not explicitly mentioned in `Scope` are not part of this assessment. Attack vectors that involve a malicious manager were not considered. The contract interacts with AaveV3 protocol which is assumed to properly function.

Tests and deployment scripts are excluded from the scope.

2.2 System Overview

This system overview describes the initially received version (Version 1) 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.

Enzyme Foundation implements a new smart account to enable Aave V3 flash loans.

2.2.1 `AaveV3FlashLoanAssetManagerLib`

`AaveV3FlashLoanAssetManagerLib` is an implementation contract that enables its `owner` to perform a `flashloan()` and implements the `executeOperation()` callback function required by Aave V3 in which the loaned funds can be used and finally paid back. It's a peripheral contract aiming to facilitate interactions of an Enzyme fund with flash loaned assets. A fund owner can add a proxy of the contract as an asset manager to their fund.

The implementation contract is configured with `_aaveReferralCode` and a `_aavePoolAddressProviderAddress` address when deployed. Furthermore, proxy contract is initialized through a call to `init()` which sets the `owner` to the caller and the `borrowedAssetsRecipient` to a provided address.

A `flashloan()` can only be performed by the `owner` which eventually calls `AaveV3Pool.flashloan()` on the Aave pool. Only a flash loan of `interestRateModes` 0 can be performed. This means that the borrowed funds must be fully repaid at the end of the transaction, including a premium. If the borrowed funds are not fully repaid the transaction will revert. `flashloan()` is called with three arguments: `_assets` (the assets to borrow), `_amounts` (the amounts to borrow of each asset) and `_encodedCalls` (the encoded calls to execute in the callback).

`executeOperation()` is the callback executed in the flash loan. It ensures that the flash loan `initiator` is the contract itself and that the caller is the configured Aave pool. Furthermore, it transfers all the borrowed assets to the `borrowedAssetsRecipient`. It then performs the encoded calls. Note that the final call should transfer back the borrowed assets and the premium to this contract address as the last step of the callback is to repay the loaned assets to the Aave pool.

2.2.2 Roles and Trust Model

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

Fund managers are meant to be the owners of `AaveV3FlashLoanAssetManagerLib`. They are the only ones that can perform a flash loan. The `borrowedAssetsRecipient` will in theory be the address of the managed vault.

The fund managers are trusted with the performed calls during the flash loan callback as they can be arbitrary.

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 our findings. The findings are split into these different categories:

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 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

6.1 `flashloan()` Is Not GSN Compatible

Informational **Version 1**

CS-SULUA3FL-001

`flashloan()` verifies that the msg.sender is the owner that initialized the contract. Therefore, this function cannot be called by a GSN relay.

6.2 `init()` Can Be Called Again if `owner` Is Zero Address

Informational **Version 1**

CS-SULUA3FL-002

If the `owner` is set to zero address, then the `init()` function can be called again by any user. This is not a security issue as if the `owner` is set to zero address it cannot call `flashloan()`. Furthermore, we assume that only a properly initialized proxy would be whitelisted to execute operations as the fund owner is trusted.