PUBLIC

Code Assessment

of the Aave V4 Tokenization Spoke Smart Contracts

February 24th, 2026

Produced for

Aave Labs

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Adam Schoeman (CISO, Aave Labs), Emilio Frangella (SVP of Engineering, Aave Labs), Stani Kulechov (CEO, Aave Labs), dear Aave Labs team,

Thank you for trusting us to help Aave Labs with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Aave V4 Tokenization Spoke according to Scope to support you in forming an opinion on their security risks.

Aave Labs implements Aave V4 Tokenization Spoke, a spoke for Aave V4 that allows minting ERC-4626 shares when supplying liquidity to the Hub.

The most critical subjects covered in our audit are solvency, ERC-4626 compliance, and Hub integration. Security regarding all the aforementioned subjects is high.

The general subjects covered are access control, signature verification, events, 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.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	1
• Code Corrected	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 following contracts are in scope:

```
src/spoke/TokenizationSpoke.sol
src/spoke/instances/TokenizationSpokeInstance.sol
```

The assessment was performed on the source code files inside the Aave V4 Tokenization Spoke 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	4th Feb 2026	4ffe2a4ed272b707faea23176d48b349afc63358	Initial Version
2	10th Feb 2026	c3b92877ec49c791e7eaaf77afe73eabd9bb7b1e	v0.5.9

For the solidity smart contracts, the compiler version `0.8.28` was chosen.

2.1.1 Excluded from scope

Any contracts not explicitly listed in the scope section above are excluded from the assessment. Third party dependencies are assumed to behave according to their specification.

Deployment scripts and tests are also 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.

The *TokenizationSpoke* is an ERC4626-compliant vault that tokenizes deposits of an asset in the Hub, allowing liquidity providers to receive fungible ERC20 share tokens that represent their claim on the underlying liquidity.

2.2.1 TokenizationSpoke

Each `TokenizationSpoke` instance is bound to a specific Hub contract and asset ID. The spoke interacts with the hub through `Hub.add()` to supply liquidity and `Hub.remove()` to withdraw. Shares minted by the Hub `add()` method go to consitute the total supply of the `TokenizationSpoke`. They are distributed to the spoke depositors according to their individual deposits.

The spoke implements the ERC4626 standard, providing familiar `deposit()`, `mint()`, `withdraw()`, and `redeem()` operations. Additionally, it supports signature-based operations through an intent system, enabling gasless transactions and meta-transaction patterns.

2.2.1.1 Spoke Operations

The spoke maintains a two-level share accounting system. At the hub level, the spoke holds hub shares obtained through calls to `Hub.add()` that deposits the underlying asset into the hub. At the spoke level, the spoke issues its own ERC20 share tokens to users in equal amounts to the the hub shares received by the spoke.

The conversion rate between spoke shares and underlying assets is derived from the Hub's share price. Users can `deposit()`/ `mint()` to obtain shares by providing assets, and `withdraw()`/ `redeem()` to receive assets by burning shares.

The spoke stores immutable references to the Hub contract, asset ID, underlying token address and decimals. The spoke's `addCap` is configured and enforced by the hub.

2.2.1.2 Intent-Based Operations

The TokenizationSpoke extends standard ERC4626 functionality with an intent system that allows users to authorize operations via EIP712 signatures. Indeed, the spoke inherits from the `IntentConsumer` abstract base contract, which provides EIP712 signature verification functionality.

The spoke supports five types of intents through signature-based functions: `depositWithSig()`, `mintWithSig()`, `withdrawWithSig()`, `redeemWithSig()`, and `permit()` for share token approvals. Each intent includes operation parameters, a keyed-nonce for replay protection, and a deadline for expiration. The `EIP712Hash` library defines the type hashes for all intent structures according to the EIP712 specification.

When a signature-based function is called, the contract verifies the deadline has not passed, constructs the EIP712 typed data hash, and validates the signature using the `SignatureChecker` library. Both EOA ECDSA signatures and ERC1271 smart contract wallet signatures are supported. The contract then consumes the nonce and executes the requested operation.

2.2.1.3 Instance Architecture

The TokenizationSpoke uses a two-contract pattern for upgradeable proxy deployments: an abstract `TokenizationSpoke` base contract containing core logic, and a concrete `TokenizationSpokeInstance` that implements the initialization function. Upgrades are tracked via `SPOKE_REVISION` using OpenZeppelin's `reinitializer` modifier.

The `SPOKE_REVISION` tracks implementation versions and may change with upgrades, and is not expected to correspond to the EIP712 domain version (initially set as `'1'`). This preserves signature validity across implementation upgrades.

2.2.1.4 Keyed-Nonce System

The `TokenizationSpoke` implements a keyed-nonce system. Nonces are stored per `(owner, key)` pair and incremented sequentially within each key, allowing users to maintain independent nonce sequences.

The `permit()` function is hardcoded to use `PERMIT_NONCE_NAMESPACE` (key `0`). The signature-based vault operations (`depositWithSig`, `mintWithSig`, `withdrawWithSig`, `redeemWithSig`) accept a packed `(key, nonce)` value chosen by the signer, permitting any `uint192` key. Users may also call `useNonce()` to invalidate the current nonce at any key.

2.3 Trust Model

The following roles exist in the TokenizationSpoke system:

- `Hub Authority`: The hub has `restricted` functionality controlled by an `authority` contract. The hub authority can configure assets, enable/disable Spokes, set interest rate strategies, and manage supply/borrow caps. The hub authority is **fully trusted**. A malicious hub authority could disable the TokenizationSpoke's access to the hub by removing it from the enabled spokes list,

preventing withdrawals and locking user funds, or add non-compliant tokens that cause accounting issues.

- `Spoke Proxy Admin`: **fully trusted**, can upgrade the TokenizationSpoke implementation to a new version if the spoke is deployed as an upgradeable proxy. A malicious proxy admin could upgrade to a malicious implementation that directly steals all user funds.
- `Users`: **untrusted**, can deposit assets and sign EIP712 intents for spoke operations (deposit, withdraw, mint, redeem, permit). Users can only authorize operations on their own assets and shares. Their signatures are cryptographically verified via EIP712 before execution.
- `Relayers/Callers`: **untrusted**, any address can submit validly signed intents or call public spoke functions on behalf of users. Relayers have no special privileges and cannot modify intent parameters or execute operations without valid signatures.

Tokens added to the hub are assumed to be standard ERC20 tokens without rebasing mechanisms, transfer fees, or transfer hooks. The hub authority is responsible for ensuring only compliant tokens are configured.

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 Open 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:

- 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 Open 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	0
Low-Severity Findings	1

- maxDeposit() Overestimates Deposit Capacity Code Corrected

6.1 `maxDeposit()` Overestimates Deposit Capacity

Correctness Low Version 1 Code Corrected

CS-AAVE4-TS-001

`maxDeposit()` calculates deposit capacity as `allowed - totalAssets()`. The `totalAssets()` function underestimates current assets because it calls `previewRedeem(totalSupply())`, which in turn calls `HUB.previewRemoveByShares()` which rounds down.

Since `totalAssets()` returns an underestimated value, subtracting it from `allowed` results in an overestimated deposit capacity.

When users attempt to deposit based on this overestimated amount, `_validateAdd()` the Hub can therefore revert.

Code corrected:

In Version 2, `maxDeposit()` was modified to compute the deposit capacity as `allowed - previewMint(totalSupply())` which underestimates the amount of assets that can be deposited in the Spoke.