PUBLIC

Code Assessment

of the Immutable aToken Vault Smart Contract

August 19, 2025

Produced for

aave

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Aave Team,

Thank you for trusting us to help Aave with this security audit.

Aave has developed the `ImmutableATokenVault`, an immutable vault for Aave's aTokens, which are interest-bearing tokens representing deposits in the Aave protocol. Previously, `ATokenVault` had been developed and audited. The scope of this audit is to review the functional equivalence of `ATokenVault` and `ImmutableATokenVault`. Hence, any issues that would have been missed in the previous audit of `ATokenVault` are out of scope for this audit.

The two contracts differ in the construction and initialization of the vault. Hence, the functional equivalence means that two vaults which have been fully initialized with the same parameters behave identically afterwards. Construction and initialization phases have been reviewed carefully. The functional equivalence holds as there were no findings.

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 `ImmutableATokenVault.sol` smart contract in the `src` folder inside the Immutable aToken Vault repository. PR #104 was used as a design specification. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	7 August 2025	44e25a15b5e173e159e574de6c38d145161f4db0	Initial Version

For the solidity smart contracts, the compiler version `0.8.22` was chosen.

The focus of this assessment is on `ImmutableATokenVault`, an immutable, i.e. non-upgradable, variant of the `ATokenVault` contract. However, the functionality should remain the same.

2.1.1 *Excluded from scope*

The correctness of `ATokenVault` is out of scope for this assessment. `ATokenVault` has previously been audited. This assessment only focuses on the equivalence of `ATokenVault` and `ImmutableATokenVault`. Furthermore, any other third-party dependencies, integrations and imports of `ATokenVault` are out of scope.

The `ATokenVault` is supposed to remain unchanged. Hence, any suggestions for the `ImmutableATokenVault` that require a change to the `ATokenVault` are out of scope.

2.1.2 *Assumptions*

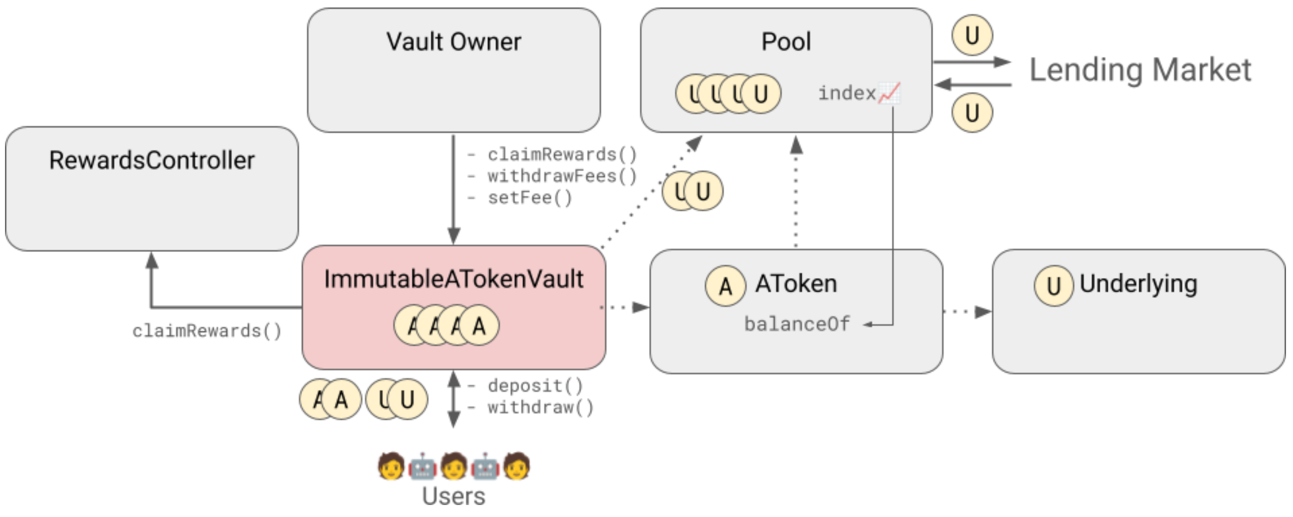
We have to make certain assumptions about the parts which are out of scope. Hence, we document that we assume the following properties hold:

- It is planned that the `ATokenVaultFactory` will be updated in the future so that it can deploy `ImmutableATokenVault` contracts. We assume that this will happen and that those changes will be correct to allow the simple deployment of the `ImmutableATokenVault` contract.
- If the underlying token would have special callbacks, e.g. ERC-777, the behavior of `ATokenVault` and `ImmutableATokenVault` could be different. However, we assume that the underlying has been carefully checked by Aave's Risk Assessment and has no special callbacks.
- If the underlying token would not handle an approval of `2**256 - 1` correctly (there are tokens that have special handling for very large approvals), there might be functional errors. Hence, we assume that the underlying approval to the pool works correctly.
- During the deployment, a `ImmutableATokenVault` can be misconfigured. This is the same for `ATokenVault`. It is assumed that malicious or misconfigured deployments have no effect as users will either check the correct configuration or use the Aave frontend where only correctly configured vaults will be shown.
- `AToken` contracts are upgradable.
  - We assume that no such upgrade breaks the functionality of the `ATokenVault`.
  - We assume that there cannot be a negative rebase.

    - We assume that the connection of 1 aToken = 1 underlying remains in future upgrades.

- We assume that during deployment the underlying is not the Vault itself. Otherwise, the behavior of `ATokenVault` and `ImmutableATokenVault` would be different. However, this is a meaningless configuration.
- When comparing the equivalence of `ATokenVault` and `ImmutableATokenVault` we assume that `ATokenVault` has been deployed and initialized atomically. This is happening atomically in the `ATokenVaultFactory` and hence a reasonable assumption. If the `ATokenVault` would not have been deployed atomically and correctly, the contracts would not be equivalent.
- When the `ATokenVault` is deployed it uses a proxy. Hence, when calling the `ATokenVault` there is also some proxy functionality available. This functionality mostly concerns the upgradability which is removed in the `ImmutableATokenVault`. To conclude, calls to `ImmutableATokenVault` and `ATokenVault` differ in the proxy behavior, but we assume that this is fine, as the point of the project is to remove the upgradability.

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.

In Aave, an `AToken` is a special, rebasing ERC-20 which is an interest-bearing for a specific underlying. The decimals of an `AToken` are the same as the decimals of the underlying. Users can obtain the `AToken` by depositing the underlying.

The `ATokenVault` allows Aave's AToken Assets to be ERC-4626 compatible. ATokens are usually rebasing, meaning that they undergo balance changes. The `ATokenVault` holds AToken and as the contract's AToken balance increases, the ERC-4626 share value grows. As such, the decimals of the `ATokenVault` are the same as the `AToken`, so the same as the underlying.

`ImmutableATokenVault` is a non-upgradeable version of `ATokenVault` that can be deployed as a standalone contract and work without a proxy. This is achieved by performing `ATokenVault's` initialization directly in the constructor of the `ImmutableATokenVault`.

List of changes done in `ImmutableATokenVault` relative to `ATokenVault`

- Constructor of `ImmutableATokenVault` receives these arguments:
    - `underlying`: The underlying ERC-20 asset address
    - `referralCode`: Aave referral code (uint16)

        - `poolAddressesProvider`: Aave v3 Pool Addresses Provider
        - `owner`: Initial owner address
        - `initialFee`: Initial fee rate as fixed point integer with 18 decimals
        - `shareName`: Name for ERC-20 vault shares and the EIP-712 signing domain
        - `shareSymbol`: ERC-20 symbol for vault shares
        - `initialLockDeposit`: Initial deposit amount of underlying ERC-20 to prevent ERC-4626 inflation attacks

    This list of arguments is a combination of the arguments of `ATokenVault.constructor()` and the arguments of `ATokenVault.initialize()`.

- The constructor of `ImmutableATokenVault` calls the `_initialize()` function. This functionality essentially contains the functionality that was previously inside the `ATokenVault.initialize()` function.
- `_disableInitializers()` has been overridden to be a no-op. This allows the `ATokenVault` constructor to be called from the `ImmutableATokenVault` constructor which can then still call `_initialize()` right afterwards.

Otherwise, `ImmutableATokenVault`, like `ATokenVault`, uses upgradeable versions of the OpenZeppelin contracts, as well as the storage layout.

2.2.1 Comparison of Deployments

Previous Deployment	New Deployment
`ATokenVault` Deployment	ERC-20 Approval to `ImmutableATokenVault`
> `ATokenVault` Constructor	`ImmutableATokenVault` Deployment
Proxy Deployment	> `ImmutableATokenVault` Constructor
> Proxy Constructor	> `ImmutableATokenVault _initialize`
ERC-20 Approval to Proxy	
`ATokenVault initialize`	

Here, we compare the previous deployment of the `ATokenVault` with the new deployment of the `ImmutableATokenVault`. We see that the previous deployment required four separate calls, while the new deployment will only require two separate calls, as the constructor directly performs the initialization.

2.3 Trust Model

Following roles exist in `ImmutableATokenVault` as in the `ATokenVault`.

- **Vault deployer**
    - Trust Level: Untrusted, since anybody can deploy a vault. Bad or malicious vaults are assumed to be ignored by the users.
    - Capabilities: Set initial parameters, fees, owner
- **Owner** (`onlyOwner` modifier):
    - Trust Level: Fully trusted
    - Capabilities: Set fee, withdraw accumulated fees, claim rewards, emergency token rescue, transfer ownership

    - Worst case impact: Can set fees up to 100%, withdraw those fees, claim all rewards, withdraw all non-aToken assets from the vault. Users would still be able to withdraw (roughly) what they have deposited.

- **End Users**:
    - Trust Level: Untrusted
    - Capabilities: Deposit, withdraw, mint, redeem operations
    - Protected by: ERC-4626 standard protections, fee calculations exclude their principal

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 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 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.

6.1 ERC-4626 Inflation Possible

Note **Version 1**

This is technically out of scope as it also concerns the original `ATokenVault`, but since the previous audit reports did not mention it, we would like to point out that:

- The `totalAssets()` can be inflated, e.g. by donating to the vault.
- Hence, return values of `convertToAssets()`, `convertToShares()`, and the return values of the `preview*()` functions can reach unexpected values. Smart contracts that build on these vaults need to handle such extreme return values correctly.

The contracts have the following protections against inflation attacks:

- During construction, an initial amount of funds is deposited. If this is done through an appropriate factory, then these can be considered dead shares.
- In the `deposit*()` functions, it is checked that the depositor receives a non-zero amount of shares.
- In the `redeem*()` functions, it is checked that the redeemer receives a non-zero amount of assets.

Overall, this means that ERC-4626 inflation is possible, but profitable attacks against simple vault depositors are not possible. However, profitable attacks might be possible against contracts building on top of the vaults if they are not carefully handling extreme values.

Lastly, griefing attacks remain possible also against simple vault depositors.

6.2 Gas Differences

Note **Version 1**

When deploying `ATokenVault` and `ImmutableATokenVault`, there are a few gas-related differences.

1. Gas costs for deployment and calls will be different, but not prohibitively different.
2. Their out-of-gas behavior is slightly different. This is not due to a programming mistake, but because the `ATokenVault` is called through a proxy. Hence, there will be nested calls which have a slightly different out-of-gas behavior than a regular call.

Overall, we can consider their functionality to be equivalent if we assume that there is always sufficient gas available when calling them.