AAVE STATA TOKEN (WATOKEN) SECURITY AUDIT REPORT

Dec 05, 2024

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals
- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

The StataToken is an EIP-4626 generic token vault/wrapper intended to be used with Aave v3 aTokens. More precisely, this is the second version of stata tokens.

The implementation is separated into two ERC20 extensions and one actual "merger" contract stitching functionality together:

1. `ERC20AaveLM` is an abstract contract implementing the forwarding of liquidity mining.
2. `ERC4626StataToken` is an abstract contract implementing the EIP-4626 methods for an underlying aToken.
3. `StataTokenV2` is the main contract inheriting `ERC20AaveLM` and `ERC4626StataToken`, while also adding `Pausability`, `Rescuability`, `Permit` and the actual initialization.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Aave DAO (Code Developed by BGD Labs)
Project name	StataTokenV2
Timeline	25.09.2024-03.12.2024
Number of Auditors	3

Project Log

Date	Commit Hash	Note
25.09.2024	f02fcb2d6ac2e08aa6bf6a8a0d38f6786cc1809a	Commit for the audit
07.11.2024	c2f6db7984a2129831eca288bab2431434ed4d7c	Commit for the re-audit
03.12.2024	665b480ab89a62b0b3b9dbab21479292b3142772	Commit with updates

Project Scope

The audit covered the following files:

File name	Link
StataTokenV2.sol	StataTokenV2.sol
ERC20AaveLMUpgradeable.sol	ERC20AaveLMUpgradeable.sol
ERC4626StataTokenUpgradeable.sol	ERC4626StataTokenUpgradeable.sol

Deployments

File name	Contract deployed on mainnet	Comment
TransparentUpgradeableProxy.sol	0x0bfc9d54Fc184518A81162F8fB99c2eACa081202	WETH
TransparentUpgradeableProxy.sol	0xD4fa2D31b7968E448877f69A96DE69f5de8cD23E	USDC
TransparentUpgradeableProxy.sol	0x7Bc3485026Ac48b6cf9BaF0A377477Fff5703Af8	USDT
TransparentUpgradeableProxy.sol	0xb80B3215EA8183a064073f9892eb64236160a4dF	USDS
StataTokenV2.sol	0x487c2C53c0866F0A73ae317bD1A28F63ADcD9aD1	
TransparentUpgradeableProxy.sol	0x775F661b0bD1739349b9A2A3EF60be277c5d2D29	wstETH
TransparentUpgradeableProxy.sol	0x0FE906e030a44eF24CA8c7dC7B7c53A6C4F00ce9	WETH
StataTokenV2.sol	0x23Db508559ca053eEE7F21b94dAC803353560B4f	
TransparentUpgradeableProxy.sol	0x57f664882F762FA37903FC864e2B633D384B411A	WETH
TransparentUpgradeableProxy.sol	0x773CDA0CADe2A3d86E6D4e30699d40bB95174ff2	wstETH
TransparentUpgradeableProxy.sol	0x7c16F0185A26Db0AE7a9377f23BC18ea7ce5d644	GNO
TransparentUpgradeableProxy.sol	0x51350d88c1bd32Cc6A79368c9Fb70373Fb71F375	USDCe
StataTokenV2.sol	0x7CB7fdeEB5E71f322F8E39Be67959C32a6A3aAA3	

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	0
Medium	0
Low	5

ID	Name	Severity	Status
L-1	Casting of `lastUpdateIndex` to uint240 despite being uint248	Low	Fixed
L-2	Unoptimized return value check	Low	Fixed
L-3	Typos in comments and documentation inaccuracies	Low	Fixed
L-4	Lack of zero address check for `INCENTIVES_CONTROLLER`	Low	Fixed
L-5	Dust when depositing AToken in the vault	Low	Fixed

1.6 Conclusion

Aave StataToken combines an ERC4626 vault with a reward distribution system. Therefore, in this audit, we paid special attention to attacks on ERC-4626 vaults and errors in rewards distribution.

Key vectors examined for the ERC-4626 contract `ERC4626StataToken`:

- We checked for rounding errors in either direction during minting and burning of assets. Everything is implemented correctly since the functions `_convertToShares()` and `_convertToAssets()` use the appropriate `Math.Rounding`.
- An inflation attack via the influence of the AToken balance on the contract is impossible since the exchange rate does not depend on it.
- We also verified the correct inheritance of the OpenZeppelin ERC-4626 contract. The functions `maxMint()`, `maxWithdraw()`, `maxDeposit()`, and `maxRedeem()` have been overridden considering that the Aave pool might be inactive, paused, frozen, or have a supply cap. The `_withdraw()` and `_deposit()` functions are also correctly overridden, except for minor potential discrepancies between the AToken balance and the underlying asset.

Key vectors tested for rewards distribution in `ERC20AaveLM`:

- Users cannot steal another user's rewards; they must be either the registered claimer or the `msg.sender` (who owns the rewards).
- If there are not enough rewards in the `StataTokenV2` contract, the user can claim part of the rewards, and the remaining amount is stored in the `unclaimedRewards` variable. Once the `StataTokenV2` contract is replenished with rewards, the user can easily claim the remaining rewards.
- We verified the correctness of tracking integral indices when initializing a new reward token, as well as during transfers, minting, and burning of shares. The tracking works correctly, with checkpoints occurring before any operations.
- Failure to accrue rewards to users due to delayed registration of a reward token. This issue exists.
- There is no possibility of a yield-stealing attack (mint and burn shares via flashloan within a single block). If a reward token is not registered and an attacker registers it in the same block as a flashloan, they still cannot steal previously unregistered rewards, as reward distribution is measured from the time of registration. This is also impossible for an already registered token, as user index checkpoints are made before any operation.
- Sending a "poisonous" token instead of a reward token to various user functions does not lead to issues, as there are checks to ensure the token is registered by the admin.
- We also verified that users can receive their pending rewards after their shares are burned.

We also went through our detailed checklist, covering other aspects such as business logic, common ERC20 issues, interactions with external contracts, integer overflows, reentrancy attacks, access control, and other potential issues.

No significant vulnerabilities were found.

Key notes and recommendations:

- The **rewards** array only has a **push** method and lacks a **pop** method. Although reward addresses can only be added by the admin in the `INCENTIVES_CONTROLLER` contract, there is a low probability of a **DoS** when `StataTokenV2._update()` is triggered if the admin adds too many reward tokens to the contract.
- Until an existing reward token is registered, rewards will accumulate but not be distributed to users. The admin can forcibly withdraw these rewards, but users may be demotivated by the possibility of missing out on a portion of their rewards.
- In case of an emergency, contracts can be **paused**, and a **rescue guardian** can transfer tokens (except ATokens backed by shares) and ETH from the contract.

2.FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

Not Found

2.4 Low

L-1	Casting of `lastUpdateIndex` to uint240 despite being uint248
Severity	Low
Status	Fixed in c2f6db79

Description

In `ERC20AaveLMUpgradeable._registerRewardToken()`, **lastUpdatedIndex** is cast to **uint240**.

ERC20AaveLMUpgradeable.sol#L302

However, its type is **uint248**.

IERC20AaveLM.sol#L12

Recommendation

We recommend casting it to **uint248**.

Client's commentary

> Fixed here PR-36

L-2	Unoptimized return value check
Severity	Low
Status	Fixed in c2f6db79

Description

In `ERC4626StataTokenUpgradeable.maxDeposit()`, when calculating the return value, **>=** should be used instead of **>**.

Currently, if **currentSupply** is greater than **supplyCap**, zero is returned. However, if these values are equal, the result is still zero.

ERC4626StataTokenUpgradeable.sol#L182

Recommendation

We recommend using **>=** instead of **>**.

Client's commentary

> Fixed here PR-36

L-3	Typos in comments and documentation inaccuracies
Severity	Low
Status	Fixed in c2f6db79

Description

- In the `_startIndex` mapping, the address is named `user` instead of `reward`.

ERC20AaveLMUpgradeable.sol#L25

- In the comments of the `ERC20AaveLMUpgradeable._getPendingRewards()` function, it states that the return value is in **WAD**, but it is actually in **asset decimals**.

ERC20AaveLMUpgradeable.sol#L198-L202
ERC20AaveLMUpgradeable.sol#L219

- In the `IERC20AaveLM` interface, the comments indicate that **rewardsIndexOnLastInteraction** and **unclaimedRewards** are in RAYs, but they are actually in asset decimals.

IERC20AaveLM.sol#L6-L7

- In the **README** file, it is stated that anyone can rescue funds using the **PermissionlessRescuable** contract. However, the code uses the **Rescuable** contract with the **onlyRescueGuardian** modifier, meaning that only a designated rescue guardian can perform rescue operations.

README.md?plain=1#L78-L79

Recommendation

We recommend correcting the typos.

Client's commentary

> Fixed here PR-36

L-4	Lack of zero address check for `INCENTIVES_CONTROLLER`
Severity	Low
Status	Fixed in c2f6db79

Description

In the constructor of **ERC20AaveLMUpgradeable**, there is no check to ensure that `INCENTIVES_CONTROLLER` is not set to the zero address. This could lead to a situation where the admin mistakenly assigns it to the zero address, rendering the contract unusable after initialization.

ERC20AaveLMUpgradeable.sol#L42

Recommendation

We recommend adding a check to ensure that **INCENTIVES_CONTROLLER** is not set to the zero address.

Client's commentary

> Fixed here PR-36

L-5	Dust when depositing AToken in the vault
Severity	Low
Status	Fixed in c2f6db79

Description

Regular users may encounter difficulties when attempting to deposit their entire AToken balance.

The deposit functions in `ERC4626StataTokenUpgradeable` transfer an exact amount of ATokens from the user, as specified in the `assets` argument:

```
function _deposit(
    address caller,
    address receiver,
    uint256 assets,
    ...
)
    ...
    SafeERC20.safeTransferFrom(
        $._aToken,
        caller,
        address(this),
        assets
    );
    ...
```

ERC4626StataTokenUpgradeable.sol#L217

However, since ATokens are rebaseable, by the time the transaction is in the mempool, the user's AToken balance might increase, leaving some dust in the user's wallet after the transfer.

Recommendation

We recommend providing the user with the option to transfer the entire available amount at the time of the transaction's execution.

Client's commentary

> Addressed here PR-36

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://twitter.com/mixbytes