certora

Security Assessment Report

aave

Clinic Steward

February-2025

Prepared for:

Aave DAO

Code developed by:

BORED
GHOSTS
DEVELOPING

Table of content

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Clinic Steward	Github Repository	3c6dfa5	EVM/Solidity 0.8

Project Overview

This document describes the security considerations for **ClinicSteward,** including all findings and recommendations reported and discussed with the developing team. The work was undertaken on **February 12, 2025**.

The following contract list is included in our scope:

- ClinicSteward.sol

Protocol Overview

The **ClinicSteward** is a permissioned helper that helps the Aave DAO repay and liquidate positions deemed bad debt. The steward was developed as part of a broader upgrade to core protocol with the introduction of pool deficit accrual upon liquidation when bad debt is generated on **Aave V3.3**. The **ClinicSteward** will be used to clean up any bad debt existing in the system that wasn't turned into a protocol deficit.

Findings Summary

The table below summarizes the review's findings, including details on type and severity.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	-	-	-
Low	3	3	3
Informational	1	1	-
Total	4		

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
		Likelihood		

Detailed Findings

ID	Title	Severity	Status
L-01	**DoS of batchRepayBadDebt by a griefer**	Low	Fixed
L-02	**DoS of batchLiquidate by a griefer**	Low	Fixed
L-03	**DoS of batchLiquidate by a griefer through budget underflow**	Low	Fixed
I-01	**Open approval of the steward to the pool**	Informational	acknowledged

Low Issues

L-01 DoS of batchRepayBadDebt by a griefer	
Severity: **Low**	Impact: **Low**

Description:
When calling batchRepayBadDebt, the function _getUsersDebtAmounts is called with the parameter usersCanHaveCollateral = false. This function performs validation to ensure that all given positions do not include collateral and reverts if any collateralized position is present.
If any user within the list has even the slightest collateral, the transaction rolls back regardless of the state of the other N positions.

For example, say the DAO passes a list of 100 positions they would like to repay. If the 100th position happens to have even as little as 1 wei of collateral, the entire execution reverts.

This behaviour opens the door for griefers to inflict damage by DoSing concrete transactions. An attacker can front-run the batchRepay transaction with a supply to one of the positions within the list (the last position in the worst case) with dust of any valid collateral on Aave. The protocol allows supplying on behalf of other users. Furthermore, upon receiving the first aToken of valid collateral, the holding address is flagged automatically as holding collateral in the Aave protocol. When the code reaches the validation of the supplied position, initially thought to be clean of collateral, the entire transaction will fail, and the gas up to this point will be lost.

Since any dust value of supply will do the trick, since we can assume an elaborate size of positions array, and since on many relevant chains, the gas cost is essentially negligible even to make thousands of supply transactions, the cost to the attacker is negligible, while the cost for the caller (the DAO) is higher by orders of magnitude.

Recommendation:
In the validation within _getUsersDebtAmounts, skip the iteration instead of reverting. Back in the batchRepayBadDebt function, when iterating over positions to repay them, skip any position with 0 debt (i.e., the positions found to have collateral).

Client's response:
The issue was addressed in the following commit PR#9

L-02 DoS of batchLiquidate by a griefer

Severity: **Low**	Impact: **Low**

Description:
When calling batchLiquidate, the function _getUsersDebtAmounts is called with the parameter usersCanHaveCollateral = true. This function does not perform any validation to check whether the position is collateralized or underwater.

In the case of liquidationCall, if the collateral asset specified by the liquidator isn't collateral of the specific position or if the health factor is above the minimum allowed, a revert will occur due to validation specified in the validateLiquidation function.

This behaviour opens the door for griefers to inflict damage by DoSing concrete transactions. An attacker can front-run the batchLiquidation transaction with a liquidation of its own to one of the positions within the list (the last position in the worst case). When the code reaches the position initially thought to be collateralized with a particular collateral, the entire transaction will revert, and the gas up to this point will be lost.
The same can be done by supplying just enough collateral to an underwater position to get it healthy again.

Recommendation:
Perform a validation in a similar manner to non-collateralized positions to prevent late failure.

Client's response:
The issue was addressed in the following commit PR#9

L-03 DoS of batchLiquidate by a griefer through budget underflow

Severity: **Low**	Impact: **Low**

Description:
When calling batchLiquidate, the function _getUsersDebtAmounts is called with the parameter usersCanHaveCollateral = true. This function does not perform any validation to check whether the position is collateralized or underwater; it only sums up the total debt of all the specified positions.

Upon pulling funds, a validation is made to ensure the maximum debt destined to be liquidated does not surpass the permitted budget approved by the DAO. If the budget is surpassed, the transaction reverts.

This behaviour opens the door for griefers to DoS liquidations by increasing the debt amount of one of the specified positions so that the total debt surpasses the permitted budget.
Since the only way to increase a position's debt is to go overwater and borrow further against healthy collateral, a user (innocent or malicious) can front-run the batchLiquidation call and supply enough collateral to borrow a hefty sum against as a legitimate position, such that the total debt will surpass the budget. This will cause a revert of the transaction.

It's important to note:

1. Since users can only borrow for themselves (there is no such thing as borrowing on behalf), the DoSer has to be included in the list of liquidatees.

2. The batchRepay function is not susceptible to this flow since _getUsersDebtAmounts validates the position has exactly 0 collateral backing it. If the position has any collateral, it will just register the value 0. Therefore, if a user will front-run to supply more collateral to later borrow against it, the position will just not be counted.

Client's response:

The issue was addressed in the following commit PR#16.

Informational Issues

I-01 Open approval of the steward to the pool

Severity: **Informational**	Impact: **None**

Description:
When calling BatchLiquidate, it's quite frequent to end up in a state where the steward's allowance to the pool is open/dangling, i.e. non-zero.

This state is feasible when liquidating a position that doesn't have enough collateral to compensate for the entire debt. Given the steward cleaning up under-the-water positions that may have been unattended for some time, this scenario is well-expected.
Calling batchLiquidate will:

1. Sum up all the debt eligible for liquidation
2. Pull this sum from the collector and approve it to the pool in preparation for the liquidation call
3. Proceed to liquidate the position.

Since the collateral isn't enough to cover the entire debt, only a portion will be liquidated, and the rest will be deemed as protocol deficit. This means that the steward pulled more debt tokens from the collector than it used for liquidation and approved the pool with more tokens than actually transferred. While the extra pulled tokens are being transferred back to the collector at the end of the function, the approval is never getting closed.

It's important to note that this state is not exploitable and that in any way, all system tokens are being cleaned up and sent back to the collector.

Example:
Say we have a single position we want to liquidate. The position is as follows (for simplicity, we assume the tokens are already converted to the same currency denominator, say USD):

- 100 debt of token A
- 20 collateral of token B and
- LB of 10% for liquidation.

Calling batchLiquidate will work in the following way:

1. When we call getDebtAmount, the total debt count is 100.
2. We pull the 100 tokens of token A from the collector to the steward and approve the pool for 100 tokens in preparation for the liquidation call.
3. Since we're in bad debt, we liquidate a sum equal to 20/LB available collateral, ~18.18 tokens, to get the entire collateral.
    a. 20 collateral goes from the liquidated user to the steward, ~18.18 debt goes from the steward to the aToken, and ~82 debt tokens are counted as deficit (the debt tokens get burnt)
4. We transfer any existing debt/collateral aTokens from the steward back to the collector.

At this point, all the tokens are where they are supposed to be, but we have an open approval of 82 debt tokens to the pool that sits hanging.

Client's response:

The issue was addressed in the following commit PR#12.

Disclaimer

The Certora Prover takes a contract and a specification as input and formally proves that the contract satisfies the specification in all scenarios. Notably, the guarantees of the Certora Prover are scoped to the provided specification and the Certora Prover does not check any cases not covered by the specification.

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.