certora

Security Assessment & Formal Verification Report

aave

Umbrella

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
Umbrella	Github	62f3850	EVM/Solidity 0.8

Project Overview

This document describes the specification and verification of **the Umbrella Slashing and Deficit Manager** which is part of the **Umbrella project** using the Certora Prover and manual code review findings. The work was undertaken from **6th of February 2025 to 13th March 2025**.

The following contract list is included in our scope:

- Umbrella.sol
- UmbrellaStkManager.sol
- UmbrellaConfiguration.sol

The Certora Prover demonstrated that the implementation of the Solidity contracts above is correct with respect to the formal rules written by the Certora team. In addition, the team performed a manual audit of all the Solidity contracts**.** During the verification process and the manual audit, **two bugs and one informational issue were discovered** - see below .

Protocol Overview

The contracts under review are part of Umbrella which is an upgraded version of the Aave Safety Module, based on a staking and slashing mechanism.
Particularly the contracts under review collectively form a framework for managing reserve deficits within the Aave ecosystem. These contracts dynamically orchestrate deficit coverage and slashing operations.

Coverage

1. We wrote several rules and invariants, and verified them formally (using Certora's prover). See a detailed description later.
2. The system's mechanisms were examined to ensure their integrity during our manual audit. Specific details regarding some of the checks performed and our thought processes are outlined below.
3. **Reserve Deficit Monitoring and Management Checks:**
    - a. Deficit Calculation Consistency
        - i. Scaled tokens price calculation.
        - ii. Rounding errors.
    - b. Successful Token Transfers
    - c. GHO-specific deficit covering
4. **Slashing Mechanism and Execution:**
    - a. Slashing only new deficit
    - b. Slashing protocol fee calculation
    - c. Price calculations
        - i. stata -atoken price calculation correctness
        - ii. deficit token price calculation
    - d. Insufficient funds in Stake Token Vault
5. **Slashing Configuration Management Checks:**
    - a. Configuration Creation & Initialization
    - b. Configuration Updating
    - c. Configuration Removal
    - d. Configuration Reinstallation
        - i. Deficit Offset and pending deficit handling
6. **Deficit Coverage and Offset Adjustment:**
    - a. Deficit Offset Management
7. **StakeToken Creation and Lifecycle Management:**
    - a. StakeToken Initialization
    - b. Lifecycle Management
        - i. Confirm that the full lifecycle of StakeTokens (creation, pausing, unpausing, emergency transfers) is managed consistently.
8. **Event Emissions**

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical			
High			
Medium	1	1	1
Low	1	1	1
Informational	2	1	1
Total	4	3	3

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
M-01	`Donation-Induced Underflow create a DoS Vulnerability`	Medium	Fixed
L-01	`Unit Mismatch in _slashAsset(..) Event Leading to Incorrect Fee Calculation`	Low	Fixed
I-01	`Unnecessary gas consumption in the removeSlashingConfigs(..)`	Informational	Fixed
I-02	`Calling the function coverPendingDeficit(.., amount) with too large “amount” might cause a revert.`	Informational	

Medium Severity Issues

M-01 Donation-Induced Underflow create a DoS Vulnerability

Severity: **Medium**	Impact: Low	Likelihood: High

Description:

The vulnerability arises from how the contract computes the effective amount of tokens used to cover a reserve’s pending deficit in the function _coverDeficit(...). When virtual accounting is active, this function calls IERC20(aToken).balanceOf(address(this)) to determine the current balance of aTokens held by the contract right after executing a safeTransferFrom(...). This balance is then used in the conditional:

Unset

```
amount = IERC20(aToken).balanceOf(address(this));
```

The intent is to account for minor rounding discrepancies when the transferred amount is slightly different from what's expected. However, if an attacker donates even a small number of aTokens directly to the contract (i.e., outside the expected transfer mechanism), the balance returned by balanceOf(address(this)) becomes inflated.

When a legitimate call is made to coverPendingDeficit(...), the function calculates the pending deficit and then calls _coverDeficit(...) to cover it. Due to the inflated balance, the effective amount determined in _coverDeficit(...) may be higher than intended. Later, when the function coverPendingDeficit(...) updates the pending deficit by executing _setPendingDeficit(...) as follows:

Unset

```
_setPendingDeficit(reserve, pendingDeficit - amount);
```

This miscalculation may result in an arithmetic underflow if the amount exceeds pendingDeficit. Such an underflow triggers a revert, effectively preventing the deficit coverage and creating a Denial-of-Service (DoS) condition for legitimate deficit cover operations.

In summary, by donating a small amount of aToken to the contract, an attacker can manipulate the balance calculation in _coverDeficit(...), triggering an underflow when _setPendingDeficit(...) is called. This flaw not only disrupts the deficit coverage process but also exposes the system to a DoS vulnerability.

BGD-labs response:
Was fixed at ab9faf29b3629ec8e53536ebfde5450a3a1a7429.

Low Severity Issues

L-01 Unit Mismatch in _slashAsset(..) Event Leading to Incorrect Fee Calculation

Severity: Low	Impact: Low	Likelihood: Medium

Description:

When_slashAsset(...) is triggered to cover a reserve's deficit by slashing staked tokens, it performs calculations using values in two different denominations. The function determines the amount of tokens to slash in the staked token units (realSlashedAmount), and separately calculates how much of the underlying asset is actually covered (newCoveredAmount) in underlying token units. The issue occurs when the function emits an event that subtracts these two values to represent the fee. Since one value is in stake-token units and the other in underlying-token units, the resulting fee is computed using mismatched denominations, leading to an incorrect fee report in the event.

BGD-labs response:
Was fixed at a4f0f4a89af9a4c79e7c4d0e6775d14230d19efe

Informational Issues

I-01 Unnecessary gas consumption in the removeSlashingConfigs(..)

Severity: **Informational**	Impact: **-**	Likelihood: **-**

Description:
In the removeSlashingConfigs(...) function, checking with map.contains(...) before calling map.remove(...) is redundant. The remove(...) method already returns a boolean indicating success, so eliminating the extra check not only simplifies the code but also reduces gas consumption by avoiding an unnecessary read operation.

BGD-labs response:
Was fixed at d8b4676c11e0b75b152c8ec31fca9d80b5b50b92

I-02 Calling the function *coverPendingDeficit(.., amount)* with too large 'amount' might cause a revert.

Severity: **Informational**	Impact: Low	Likelihood: Low

Description:
The function *coverPendingDeficit(reserve,amount)* is called by an AIP some time after a slashing is performed. Its role is to transfer money from its caller (the *COVERAGE_MANAGER_ROLE*) to the pool, where the amount of the money is the minimum between the *pending-deficit* and the parameter "*amount*" (parameter of the function *coverPendingDeficit(..)*). Since that function is called after slashing was performed, it is

guaranteed that the balance of *COVERAGE_MANAGER_ROLE* will be at least as the *pending-deficit*, and usually the function won't revert even if the "amount" parameter is too large.

Although usually it will run fine, a problem might arise in the case that the parameter "amount" is by mistake too large. An attacker can perform a front-run and call to *slash()* before the call to *coverPendingDeficit()*. The call to *slash()* might increase the value of *pending-deficit*, and if the minimum between *pending-deficit* and the "*amount*" parameter is larger than the balance of the *COVERAGE_MANAGER_ROLE*, the call to *coverPendingDeficit(..)* will revert.

BGD-labs response:
We plan to specify the exact amount to cover the deficit.

Formal Verification

Verification Notations

Formally Verified	The rule is verified for every state of the contract(s), under the assumptions of the scope/requirements in the rule.
Formally Verified After Fix	The rule was violated due to an issue in the code and was successfully verified after fixing the issue
Violated	A counter-example exists that violates one of the assertions of the rule.

General Information

Global assumptions: The message sender is not any of the involved contracts (specifically it is not the Umbrella, nor the StakeToken).
Solidity version: solc8.27.
Loop unrolling: 2.
Links: A link to Certora's prover report can be found here (the rules) or here (the invariant).

Formal Verification Properties

In the tables below we specify all the formally verified rules that we wrote for the verification of the Umbrella, and give a detailed description for them.

P-01. Integrity of slashing

Status: Verified	Property Assumptions:

Rule Name	Status	Description	Rule Assumptions
integrity_of_slashing	Verified	*We check the following aspects of slash(..):* *1. The slashed-amount can't exceed the value of: real-deficit - deficit-offset - pending-deficit.* *2. The pending-deficit is updated correctly.* *3. The actual-slashed-amount can't exceed the deficit that can be covered + fees.* *4. The balance of the contract grows as expected.*	

P-02. Integrity of coverPendingDeficit

Status: Verified	Property Assumptions:

Rule Name	Status	Description	Rule Assumptions
integrity_of_coverPendingDeficit	Verified	*We check the following aspects of coverPendingDeficit():* *1. The covered-amount (which is the value returned from the function) equals to the minimum between the pending-deficit and amount (a parameter passed to the function).* *2. The pending-deficit is updated correctly.* *3. The POOL's function, eliminateReserveDeficit(), is called with the correct parameters.* *4. The correct amount of money is transferred to the Umbrella contract (either from the AToken or from the reserve depending whether virtual accounting is enabled or not). This money is transferred to the pool in the function eliminateReserveDeficit(), but we don't check it.*	

P-03. Integrity of coverDeficitOffset

Status: Verified	Property Assumptions:

Rule Name	Status	Description	Rule Assumptions
integrity_of_coverDeficitOffset	Verified	*We check the following aspects of coverDeficitOffset():* *1. The pending-deficit value isn't changed.* *2. The covered-amount (which is the value returned from the function), plus pending-deficit can't exceed the deficit of the asset. Moreover, covered-amount can't exceed the offset-deficit or the amount (a parameter passed to the function).* *3. The deficit-offset is updated correctly.* *4. The POOL's function, eliminateReserveDeficit(), is called with the correct parameters.* *5. The correct amount of money is transferred to the Umbrella contract (either from the AToken or from the reserve depending whether virtual accounting is enabled or not).This money is transferred to* *the pool in the function eliminateReserveDeficit(), but we don't check it.*	

P-04. The possible-slashing-amount can't be changed

Status: Verified	Property Assumptions:

Rule Name	Status	Description	Rule Assumptions
possible_slashing_amount_cant_be_changed	Verified	*Define the possible-slashing-amount to be:* *deficit-in-pool - pending-deficit - deficit-offset.* *We verify that the possible-slashing-amount does not change except for the following functions that obviously should change it:* *- setDeficitOffset(..): this function simply changes the deficit-offset.*	

		- slash(..): this function changes the pending-deficit. *- coverReserveDeficit(..): this function changes the deficit-in-pool.* *- updateSlashingConfigs(..): this function changes the deficit-offset.*	

P-05. The pending-deficit can't exceed the real-deficit

Status: Verified	Property Assumptions:

Rule Name	Status	Description	Rule Assumptions
pending_deficit_cant_exceed_real_deficit	Verified	*An invariant:* *Any operation of the Umbrella contract can't make the pending-deficit bigger than the real-deficit.*	

P-06. slashing can't DOS other functions

Status: Verified	Property Assumptions:

Rule Name	Status	Description	Rule Assumptions
slashing_cant_DOS_other_functions	Verified	*The rule checks that an attacker can't use the slash function in order to make other functions to revert. That is, if a function f does not revert when executed from some state, then if an attacker front-run the slash() function (from the same state) then f still doesn't revert.* ***Note**: We only check the slash() function because it is the only non-view function that can be run without a special role.*	

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.