Morpho Aave V2 Security Analysis

by Pessimistic

This report is private

July 4, 2022

Abstract

In this report, we consider the security of smart contracts of Morpho Aave V2 project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. A single audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, a security audit is not investment advice.

Summary

In this report, we considered the security of Morpho Aave V2 smart contracts. We performed our audit according to the procedure described below.

The audit showed Stuck tokens critical issue and several issues of medium severity, including Excessive rewards, Possible DoS, ERC20 standard violation, and Overpowered owner. Moreover, several low-severity issues were found.

The overall code quality is good. The project has a yellow paper; the codebase has detailed NatSpec comments.

General recommendations

We recommend fixing the mentioned issues.

Project overview

Project description

For the audit, we were provided with Morpho Aave V2 project on a private GitHub repository, commit 41789b4004b2391cf28d2741840beab7ae3696e8.

The scope of the audit included:

- **contracts/aave-v2/interfaces/** folder
- **contracts/aave-v2/rewards-managers/** folder
- **contracts/aave-v2/EntryPositionsManager.sol** file
- **contracts/aave-v2/ExitPositionsManager.sol** file
- **contracts/aave-v2/InterestRatesManager.sol** file
- **contracts/aave-v2/MatchingEngine.sol** file
- **contracts/aave-v2/Morpho.sol** file
- **contracts/aave-v2/MorphoGovernance.sol** file
- **contracts/aave-v2/MorphoStorage.sol** file
- **contracts/aave-v2/MorphoUtils.sol** file
- **contracts/aave-v2/PositionsManagerUtils.sol** file

The documentation for the project included **Yellow_Paper.pdf** file (v0.5), sha1sum `3da685a988dce07785d1f1de00958153de0ea81d`. The codebase is covered with detailed NatSpec comments.

All 155 unit tests and 17 fuzzing tests pass successfully. The code coverage is not measured.

The total LOC of audited sources is 2160.

Procedure

In our audit, we consider the following crucial features of the code:

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- Automated analysis
  - We scan the project's codebase with the automated tool Slither.
  - We manually verify (reject or confirm) all the issues found by the tool.
- Manual audit
  - We manually analyze the codebase for security vulnerabilities.
  - We assess the overall project structure and quality.
- Report
  - We reflect all the gathered information in the report.

Inter alia, we verify that:

- Morpho protocol properly implements documented functionality, especially for supply, borrow, withdraw, liquidate and repay actions.
- Contracts interact properly, and the value exchange process goes as expected when users enter or exit the protocol.
- Morpho protocol correctly integrates with AAVE v2 protocol.
- The project is resistant to reentrancy, flashloan and front-running attacks.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

C01. Stuck tokens (out of scope)

The **IncentivesVault** contract blocks `rewardToken`s without any possibility of retrieving them.

When users call the `claimRewards` function of **Morpho** contract with the `_tradeForMorphoToken` argument set to `true`, the contract transfers `rewardToken`s to **IncentivesVault** contract at lines 195–199. After this, **Morpho** contract calls the `tradeRewardTokensForMorphoTokens` function of **IncentivesVault** contract that recalculates the reward into `morphoToken`s and transfers them to the user. However, **IncentivesVault** contract does not allow transferring `rewardToken`s any further. Thus, these tokens remain on the contract forever.

Consider adding functionality for transferring `rewardToken`s from **IncentivesVault** contract.

Medium severity issues

Medium issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. Excessive rewards

Morpho contract calls `claimRewards` function of **IncentivesController** contract from AAVE protocol with `amountOfRewards` argument. This function returns the `amount` of tokens to transfer from **IncentivesController** contract. The function may return a smaller value than the requested `amountOfRewards` due to the condition at line 192. However, further, at line 200, **Morpho** contract calls `tradeRewardTokensForMorphoTokens` function of **IncentivesVault** contract with the initial `amountOfRewards` value. As a result, the system might use more tokens for the reward than it receives from AAVE protocol.

Consider saving the returned value from `IncentivesController.claimRewards` call and passing it to the `tradeRewardTokensForMorphoTokens` function of **IncentivesVault** contract at line 200 of **Morpho** contract.

M02. Possible DoS

In **PositionsManagerUtils** contract, `_underlyingToken.safeApprove` calls at lines 55 and 104 can result in denial of service when working with `USDT`. Due to the check at line 205, `USDT` contract will revert on the second call. For proper work, `USDT` expects the consequent calls of `approve` function with zero and non-zero values, respectively.

M03. ERC20 standard violation (out of scope)

ERC-20 standard states:

```
Callers MUST handle false from returns (bool success). Callers MUST
NOT assume that false is never returned!
```

However, the `tradeRewardTokensForMorphoTokens` function of **IncentivesVault** contract does not check the returned value from the `transfer` call at line 137.

M04. Overpowered owner

The owner of **MorphoGovernance** contract has excessive powers. Inter alia, the owner can change the `entryPositionsManager`, `exitPositionsManager`, `rewardsManager`, and `treasuryVault` addresses, which allows the owner to gain control over users' funds.

In the current implementation, the system depends heavily on the owner of the contract. Thus, there are scenarios that can lead to undesirable consequences for the project and its users, e.g., if the owner's private keys become compromised.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Function visibility (out of scope)

Consider declaring the `calculateCompoundedInterest` function of **Lens** contract as `external` instead of `public` to improve code readability and optimize gas consumption.

L02. Rounding issue (out of scope)

In **Lens** contract, calculations at lines 190, 193, 234, 239, 241, and 244 will decrease the amount of tokens by one in most cases due to the rounding issue. We recommend implementing proper logic to consider fractions when calculating tokens.

L03. No version control

In **MorphoGovernance** contract, the `initialize` function does not provide any functionality to set the version of the contract. Since the contract works via a proxy contract, the lack of versioning might result in issues with project maintenance.

L04. Gas consumption

In **ExitPositionsManager** contract, the `_safeWithdrawLogic` function will call the `_updateSupplierInDS` function twice if both `onPoolSupply` and `remainingToWithdraw` fields of `vars` struct are greater than `0`. Consider moving the call at line 279 inside the `if` block at lines 281–300.

L05. Gas quota exceedance

In **ExitPositionsManager** contract, the `_safeWithdrawLogic` function allows spending a double quota of gas due to a bug at line 361. The function should call the `_unmatchBorrowers` function with `vars.maxGasForMatching` value as a third argument instead of `_maxGasForMatching`.

L06. Missing inheritance (out of scope)

SwapManagerUniV2 contract should implement `IOracle` interface.

L07. Redundant check

In **ExitPositionManager** contract, the `_getUserHealthFactor` function handles the case of zero debt at line 595. Thus, the debt cannot be zero after this check. Therefore, the check at line 640 is redundant.

Moreover, the contract calls the `_getUserHealthFactor` function only from `_withdrawAllowed` and `_liquidationAllowed` internal functions. These functions, in turn, are only called when a user has a debt due to checks at lines 141 and 185 of **ExitPositionManager** contract. These checks make the condition at line 595 redundant.

L08. Functionality duplication

The `_repayToPool` function of **PositionsManagerUtils** contract calculates the amount to repay. This functionality duplicates the `repay` function of **LendingPool** contract of AAVE v2 protocol. In this case, the only check that Morpho protocol should perform is to verify that Morpho has variable debt tokens on its balance (for the case of repay on behalf) since AAVE reverts on zero amount repay. Consider omitting the amount correction on Morpho side and using the AAVE repay amount instead.

L09. Variable shadowing

In **InterestRatesManager** contract, the `poolIndexes` variable duplicates the name of the storage variable at line 57. Consider renaming it to improve code readability and avoid possible confusion.

L10. Protocols misalignment

In AAVE protocol, the `repay` function of **LendingPool** contract returns the repaid amount at line 289. However, the `_repayToPool` function of **PositionsManagerUtils** contract ignores the returned value from `LendingPool.repay` call, which equals to the variable debt tokens balance of **Morpho** contract on AAVE protocol. Thus, the function does not consider the case when the repaid amount differs from the `_amount` argument. This might result in misalignment of token amounts calculation between the protocols in:

1. **EntryPositionsManager** contract at line 139.
2. **ExitPositionsManager** contract at line 438.
3. **ExitPositionsManager** contract at line 529.

Notes

N01. Position liquidation risk

In case of high price volatility, a third party might liquidate the Morpho position in AAVE before users' positions are liquidated. In this case, inner accounting (`onPool` value) of users' balances becomes obsolete. The protocol does not consider such an event. Thus, the system becomes vulnerable to outer liquidation.

This analysis was performed by Pessimistic:

Vladimir Tarasov, Security Engineer
Vladimir Pomogalov, Security Engineer
Evgeny Marchenko, Senior Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

July 4, 2022