Aave

Aave Protocol v3

Smart Contract Security Assessment Report

Version: 2.0

June, 2023

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Aave Protocol v3 smart contracts. The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the Aave Protocol v3 smart contracts contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Aave Protocol v3 smart contracts.

Overview

Aave is a platform that permits users to lend and borrow tokens (and Ether) on the Ethereum blockchain. The contracts reviewed in this report constitute version 3 of the Aave protocol. The version 3 contracts provide a number of improvements and extra features to their version 2 predecessors. Some of the core updates include:

- **eMode -** improves the capital efficiency of users by categorising assets. Users in eMode will only be allowed to borrow assets in that category but will receive improved LTV and liquidation threshold for assets supplied in that category.
- **Isolation Mode -** reduces the risk on certain assets by adding a debt ceiling which limits the amount that can be borrowed for an asset.
- **Portal -** allows for ATokens to be transferred between different chains through a bridge.

Security Assessment Summary

This review was conducted on the files hosted on the Aave Protocol v3 and were initially assessed at commit e76882a then reviewed again at commit 5ea9ad8.

Note: the OpenZeppelin libraries and dependencies were excluded from the scope of this assessment.

The manual code review section of the report, focused on identifying any and all issues/vulnerabilities associated with the business logic implementation of the contracts. Specifically, their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout). Additionally, the manual review process focused on all known Solidity anti-patterns and attack vectors. These include, but are not limited to, the following vectors: re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers. For a more thorough, but non-exhaustive list of examined vectors, see [**?**, **?**].

To support this review, the testing team used the following automated testing tools:

- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`
- Surya: `https://github.com/ConsenSys/surya`

Output for these automated tools is available upon request.

Findings Summary

The testing team identified a total of 12 issues during this assessment. Categorized by their severity:

- Low: 3 issues.
- Informational: 9 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Aave Protocol v3 smart contracts. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as “informational”.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
AAV3-01	`setEModeCategory()` Invalid Parameters	Low	Resolved
AAV3-02	Division by Zero Returns Zero	Low	Resolved
AAV3-03	User's Borrowing Status is not Updated When Their Entire Debt is Liquidated	Low	Resolved
AAV3-04	Verification `POOL` Matches the Configurator	Informational	Resolved
AAV3-05	Event `MintUnbacked` Field `user` is Always Set to Bridge	Informational	Closed
AAV3-06	No Getter For `_stableRateExcessOffset`	Informational	Resolved
AAV3-07	Unused Struct Variables and Errors	Informational	Resolved
AAV3-08	Configurator Decimals Are Parameters Rather than Fetched	Informational	Closed
AAV3-09	Repeated Arguments Passed in `validateRebalanceStableBorrowRate()`	Informational	Resolved
AAV3-10	Gas Optimisation - Reduce Ternary Operators	Informational	Resolved
AAV3-11	Unnecessary Storage Reads When Fetch Debt	Informational	Resolved
AAV3-12	Miscellaneous General Comments	Informational	Resolved

AAV3-01	`setEModeCategory()` Invalid Parameters		
Asset	`PoolConfigurator.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The function `setEModeCategory()` takes the parameters `ltv`, `liquidationThreshold`, `liquidationBonus`. These parameters are used when a user supplies collateral in eMode with a matching eMode category.

There is an assertion when configuring a reserve as collateral that if `liquidationThreshold == 0` then `liquidationBonus == 0`.

This condition is not enforced in `setEModeCategory()`, instead the requirement exists that `liquidationBonus > 100%`. Breaching this condition may result in a liquidation bonus which is greater than the balance of the user's collateral.

This issue is further inflated by the lack of `_checkNoSuppliers()` for ATokens, which if this check was present it would prevent zeroing values while there is a supply of ATokens. Thus, users may currently be using these eMode parameters as part of their collateral which if zeroed will essentially remove their collateral exposing them to liquidation.

Another related issue is that it is possible to call `setEModeCategory()` after `setAssetEModeCategory()` to reduce the `ltv` and `liquidationThreshold` to below that of the assets in this category. Since the eMode variables will always be used over the individual asset variables, if a user is in eMode it is possible to reduce the borrowing power of an asset by calling `setEModeCategory()`.

Recommendations

Consider handling the case where an eMode category `ltv`, and `liquidationThreshold` are being zeroed.

A potential solution is to prevent setting these variables to zero. To disable this category the function `setAssetEModeCategory()` may be used to remove all assets from this category.

To prevent reducing the `ltv` and `liquidationThreshold` of an eMode category below the asset's parameters requires iterating over all assets and for those with a matching eMode category ensuring the new parameters are below each asset's parameters.

Resolution

Pull request #592 resolves this issue by implementing the recommendations. The case where `ltv` and `liquidationThreshold` are set to zero is handled by reverting. Additionally, the function

`setEModeCategory()` now iterates through all assets and for those with the same eMode ID, ensures that both the LTV and liquidation threshold are less than those of the eMode category.

AAV3-02	Division by Zero Returns Zero		
Asset	`WadRayMath.sol` & `PercentageMath.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The math libraries `WadRayMath` and `PercentageMath` perform operations on `uint256` with a set number of decimal places. The following functions provide division operations accounting for the decimal places:

- `WadRayMath.rayDiv()`
- `WadRayMath.wadDiv()`
- `PercentageMath.percentDiv()`

The functions have been optimised to use a minimal amount of gas by using the assembly `div` operation. However this operation will return zero when the denominator is zero rather than reverting as the normal solidity `/` operation does. As a result each of the functions listed above will instead return zero rather than reverting when the denominator is zero.

Recommendations

We recommend updating the code to revert when the denominator is zero rather than returning zero.

Resolution

The library `WadRayMath` was updated to revert when the denominator is zero in PR#356.

The library `PercentageMath` was updated to revert when the denominator is zero in PR#364.

AAV3-03	User's Borrowing Status is not Updated When Their Entire Debt is Liquidated		
Asset	`LiquidationLogic.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

When a user has a position that no longer satisfies the required ratio of collateral-to-debt, that user may be liquidated via the function `liquidationCall()`. If the user's position breaches a certain threshold the entire position may be liquidated in one transaction.

If the entire debt of a user is liquidated for an asset, the user should no longer be considered borrowing. However the function `executeLiquidationCall()` will not change the borrowing status for a user.

The impact is that a user may have no debt for an asset but still be marked as borrowing if their entire debt is liquidated.

Recommendations

We recommend updating a user's borrowing status to be `false` for the asset if their entire debt balance was liquidated.

Resolution

The issue has been resolved in PR#556 by adding the following statement, which checks if the entire debt balance is liquidated.

```
if (vars.userTotalDebt == vars.actualDebtToLiquidate) {
    userConfig.setBorrowing(debtReserve.id, false);
}
```

AAV3-04	Verification `POOL` Matches the Configurator
Asset	`AToken.sol`, `StableDebtToken.sol` & `StableDebtToken.sol`
Status	**Resolved:** See Recommendations
Rating	Informational

Description

The `PoolConfigurator` can be used to add new assets to the protocol. When new assets are added, they require an implementation of `AToken`, `StableDebtToken` and `VariableDebtToken`. Each of these contracts have the address of the `Pool` as a parameter in their constructor.

When the `PoolConfigurator` initialises the contracts, via the function `initReserves()`, a proxy is setup for each of `AToken`, `StableDebtToken` and `VariableDebtToken`.

There are no checks during the initialisation stage to ensure that the `PoolConfigurator._pool` matches the `POOL` passed to the constructor of the implementation. Any errors here would result in a misconfiguration and potential loss of user funds if they attempted to deposit into the protocol.

Recommendations

Consider passing `PoolConfigurator._pool` to the `initialize()` function of each of `AToken`, `StableDebtToken` and `VariableDebtToken` and ensure it matches the `POOL` variable stored in these contracts.

Recommendations

The recommendation has been implemented in PR #603.

AAV3-05	Event `MintUnbacked` Field `user` is Always Set to Bridge
Asset	`BridgeLogic.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The event `MintUnbacked` contains the field `user`. This field is set in `BridgeLogic.executeMintUnbacked()` to be `msg.sender`.

However since this function can only be called by the bridge, `msg.sender` will always be the bridge.

Recommendations

Consider changing the field name to `bridge` rather than `user`.

Alternatively, consider adding `user` as a paramater to the function `mintUnbacked()` which may be passed from the bridge and thus emitted correctly in the event.

Resolution

The development team has marked this issue as *won't fix* in accordance with the comment below.

The bridge is a role and can be held by multiple addresses so there may be different bridges that are minting. Thereby multiple users of the function. The actor receiving the funds (user in issue?), is already a parameter of `mintUnbacked()` as it is the `onBehalfOf` that will receive the funds.

AAV3-06	No Getter For `_stableRateExcessOffset`
Asset	`DefaultReserveInterestRateStrategy.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The state variable `_stableRateExcessOffset` does not have a getter method. As a result it may be challenging for users to work out future interest rates.

Recommendations

Consider adding a getter for this variable to `DefaultReserveInterestRateStrategy`.

Resolution

A getter `getStableRateExcessOffset()` has been added in PR #583.

AAV3-07	Unused Struct Variables and Errors
Asset	`contracts/*`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The following is a list of variables that are either unused or are set but never read:

- `ValidationLogic.sol`:
  - `ValidateBorrowLocalVars.currentLiquidationThreshold`
- `DataTypes.sol`:
  - `FinalizeTransferParams.toEModeCategory`
- `FlashLoanLogic.sol`:
  - `FlashLoanLocalVars.debtToken`
- `ConfiguratorInputTypes.sol`:
  - `InitReserveInput.underlyingAssetName`

In addition to unused struct variables there is also one unused error in `Errors.sol` and that is `FLASHLOAN_PREMIUMS_MISMATCH`.

Recommendations

The unused struct variables and error may be safely deleted.

Resolution

The unused variables and error have been removed in PR #587.

AAV3-08	Configurator Decimals Are Parameters Rather than Fetched
Asset	`ConfiguratorLogic.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

Setting up reserves involves initialising required contracts; `AToken`, `StabledDebtToken` and `VariableDebtToken`. Each of these tokens are ERC20 derivatives and require `decimals` to be passed as an argument.

The value of the variable `decimals` is passed as a parameter to `PoolConfigurator.initReserves()`. There is the assumption this value should match the decimals of the underlying asset. However there are no checks to ensure this assumption holds.

The impact would be a misconfiguration of the deployment which could potentially go undetected until users are over or under rewarded for their transactions.

Recommendations

This assumption can be enforced by instead fetching the `decimals` from the underlying asset using the ERC20 `decimals()` function.

Resolution

The development team has opted not to implemement the recommendation. They have reasoned that since `decimals()` is an optional ERC20 function there is no gaurantee that all tokens will implement this and hence it will be passed as a constructor parameter rather than fetched. The full comment can be seen in this issue.

AAV3-09	Repeated Arguments Passed in `validateRebalanceStableBorrowRate()`
Asset	`ValidationLogic.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The function `validateRebalanceStableBorrowRate()` takes the arguments `reserveCache`, `stableDebtToken`, `variableDebtToken`, and `aTokenAddress`.

This is unnecessary as `reserveCache` contains each of the token addresses and thus do not need to be passed separately.

The following code snippet shows the repeated arguement passing.

```
IERC20 stableDebtToken = IERC20(reserveCache.stableDebtTokenAddress);
IERC20 variableDebtToken = IERC20(reserveCache.variableDebtTokenAddress);
uint256 stableDebt = IERC20(stableDebtToken).balanceOf(user);

ValidationLogic.validateRebalanceStableBorrowRate(
  reserve,
  reserveCache,
  asset,
  stableDebtToken,
  variableDebtToken,
  reserveCache.aTokenAddress
);
```

Recommendations

We recommend removing the parameters `stableDebtToken`, `variableDebtToken`, and `aTokenAddress` from `validateRebalanceStableBorrowRate()` and instead extracting them from `reserveCache` when they are required.

Resolution

The recommendation has been implemented in PR #599 by removing the extra parameters.

AAV3-10	Gas Optimisation - Reduce Ternary Operators
Asset	`DefaultReserveInterestRateStrategy.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

In the function `calculateInterestRates()` there are multiple ternary operators that operate over the same condition. That is if `totalDebt` is greater than or less than zero. This can be seen in the following code snippet:

```
vars.stableToTotalDebtRatio = vars.totalDebt > 0
  ? params.totalStableDebt.rayDiv(vars.totalDebt)
  : 0;

vars.currentLiquidityRate = 0;
vars.currentVariableBorrowRate = _baseVariableBorrowRate;
vars.currentStableBorrowRate = getBaseStableBorrowRate();

vars.borrowUsageRatio = vars.totalDebt == 0
  ? 0
  : vars.totalDebt.rayDiv(vars.availableLiquidity + vars.totalDebt);

vars.supplyUsageRatio = vars.totalDebt == 0
  ? 0
  : vars.totalDebt.rayDiv(vars.availableLiquidity + params.unbacked + vars.totalDebt);
```

Recommendations

Theses ternary operators can be combined into a single if-statement as follows.

```
if (vars.totalDebt > 0) {
    vars.stableToTotalDebtRatio = params.totalStableDebt.rayDiv(vars.totalDebt);
    vars.borrowUsageRatio = vars.totalDebt.rayDiv(vars.availableLiquidity + vars.totalDebt);
    vars.supplyUsageRatio = vars.totalDebt.rayDiv(vars.availableLiquidity
        + params.unbacked + vars.totalDebt);
}
```

The gas saving is only 77 gas but may also reduce code complexity.

Resolution

The gas saving has been implemented in PR #581.

AAV3-11	Unnecessary Storage Reads When Fetch Debt
Asset	`BorrowLogic.sol` & `LiquidationLogic.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

To measure user debt levels, both the stable debt and variable debt external calls must be made to their respective contracts.

A helper function `Helpers.getUserCurrentDebt()` is provided to assist with these external calls. The function takes a `reserve` as storage and reads both the `stableDebtTokenAddress` and `variableDebtTokenAddress` from storage before making the required contract calls.

In each case where this helper is used `reserve.cache()` has already been called, thus `stableDebtTokenAddress` and `variableDebtTokenAddress` are already stored in memory. As a result we can save a storage load by using the token addresses from memory in `reserveCache` rather than storage.

This pattern occurs three times in the following functions:

- `BorrowLogic.executeSwapBorrowRateMode()`
- `BorrowLogic.executeRepay()`
- `LiquidationLogic.liquidationCall()`

Recommendations

Consider either removing the function `getUserCurrentDebt()` and fetch the debt manually at each location.

Alternatively consider updating the parameters of the function to instead take `stableDebtTokenAddress` and `variableDebtTokenAddress` rather than `reserve`.

Resolution

The gas optimisation has been implemented in PR #588.

AAV3-12	Miscellaneous General Comments
Asset	`contracts/`
Status	**Resolved:** See Resolution
Rating	Informational

Description

This section describes general observations made by the testing team during this assessment that do not have direct security implications:

- **Inconsistent use `a != 0` and `a > 0` for `uint256`.**
  These two statements are equivalent for `uint` values since they cannot be less than zero. Consider updating all occurrences to `a != 0`.
- **Ensure all revision numbers are updated before deployment.**
  For example `Pool.POOL_REVISION` is currently 2 matching the previous version. Ensure each revision is updated before deployment.
- `CONFIGURATOR_REVISION` **is** `internal` **when all other equivalent revision variables are** `public`**.**
  Consider changing `CONFIGURATOR_REVISION` in `PoolCongifurator.sol` to `public` visibility to align with the other contracts.
- `executeMintUnbacked()` **has unnecessary parameters.**
  The function `executeMintUnbacked()` in `BridgeLogic.sol` takes parameters `reserves`, `reserve` and `asset`.
  However `reserve = reserves[asset]` and thus we can reduce the number of parameters by removing `reserve`.
- **Inconsistent use of** `from` **vs** `user` **in events.**
  The following objects use variable name `from`:
  - `IScaledBalanceToken.sol` event `Burn`
  - `IAToken.sol` function `burn()`

  The remaining burn and mint functions instead have `user`. Consider updating all occurrences to match.
- `LiquidationLogic.sol` **typo on line [92–93].**
  "... receives a proportionally amount of the 'collateralAsset' ..." -> "proportional"
- **Use of** `Atoken` **rather than** `AToken` **in** `LiquidationLogic.sol`**.**
  We recommend updating all occurrences of `Atoken` to `AToken`.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The development team have acknowledged these findings, addressing all issues in the following PR #586.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are provided alongside this document. The `brownie` framework was used to perform these tests and the output is given below.

```
test_borrow                                                PASSED  [0%]
test_borrow_isolation_mode                                 PASSED  [1%]
test_repay                                                 PASSED  [2%]
test_repay_isolation_mode                                  PASSED  [2%]
test_rebalance_stable_borrow_rate                          PASSED  [3%]
test_swap_borrow_rate_mode_stable                          PASSED  [4%]
test_swap_borrow_rate_mode_variable                        PASSED  [4%]
test_mint_unbacked                                         PASSED  [5%]
test_mint_unbacked_supply_and_borrows                      PASSED  [6%]
test_mint_unbacked_cap                                     PASSED  [6%]
test_mint_unbacked_zero                                    PASSED  [7%]
test_only_bridge_modifier                                  PASSED  [8%]
test_back_unbacked                                         PASSED  [9%]
test_back_unbacked_erc20_fails                             PASSED  [9%]
test_set_user_emode                                        PASSED  [10%]
test_flashloan_repay                                       PASSED  [11%]
test_flashloan_borrow                                      PASSED  [11%]
test_flashloan_simple                                      PASSED  [12%]
test_liquidation_call                                      PASSED  [13%]
test_linear_interest                                       PASSED  [13%]
test_linear_interest_one_year                              PASSED  [14%]
test_linear_interest_max_values                            PASSED  [15%]
test_linear_interest_zero                                  PASSED  [15%]
test_compound_interest                                     PASSED  [16%]
test_compound_interest_overflows                           PASSED  [17%]
test_compound_interest_zero                                PASSED  [18%]
test_percent_mul                                           PASSED  [18%]
test_percent_mul_zero                                      PASSED  [19%]
test_percent_mul_overflow                                  PASSED  [20%]
test_percent_div                                           PASSED  [20%]
test_percent_div_zero                                      PASSED  [21%]
test_percent_div_overflow                                  PASSED  [22%]
test_init_reserves                                         PASSED  [22%]
test_drop_reserve                                          PASSED  [23%]
test_update_atoken                                         PASSED  [24%]
test_update_stable_debt_token                              PASSED  [25%]
test_update_variable_debt_token                            PASSED  [25%]
test_set_reserve_borrowing                                 PASSED  [26%]
test_configure_reserve_as_collateral                       PASSED  [27%]
test_configure_reserve_as_collateral_large_ltv             PASSED  [27%]
test_configure_reserve_as_collateral_non_zero_bonus        PASSED  [28%]
test_configure_reserve_as_collateral_excessive_bonus       PASSED  [29%]
test_configure_reserve_as_collateral_small_bonus           PASSED  [29%]
test_configure_reserve_as_collateral_supply                PASSED  [30%]
test_set_reserve_stable_rate_borrowing                     PASSED  [31%]
test_set_reserve_active                                    PASSED  [31%]
test_set_reserve_active_atokens                            PASSED  [32%]
test_set_reserve_freeze                                    PASSED  [33%]
test_set_borrowable_in_isolation                           PASSED  [34%]
test_set_reserve_pause                                     PASSED  [34%]
test_set_reserve_factor                                    PASSED  [35%]
test_set_debt_ceiling                                      PASSED  [36%]
```

```
test_set_borrow_cap                                          PASSED  [36%]
test_set_supply_cap                                          PASSED  [37%]
test_set_liquidation_protocol_fee                            PASSED  [38%]
test_set_emode_category                                      PASSED  [38%]
test_set_emode_category_invalid_cases                        XFAIL   [39%]
test_set_emode_category_below_asset                          XFAIL   [40%]
test_set_asset_emode_category                                PASSED  [40%]
test_set_asset_emode_category_invalid_threshold              PASSED  [41%]
test_set_unbacked_mint_cap                                   PASSED  [42%]
test_set_pool_paused                                         PASSED  [43%]
test_update_bridge_protocol_fee                              PASSED  [43%]
test_update_flash_loan_premium_total                         PASSED  [44%]
test_update_flash_loan_premium_to_protocol                   PASSED  [45%]
test_only_pool_admin                                         PASSED  [45%]
test_only_emergency_admin                                    PASSED  [46%]
test_only_emergency_or_pool_admin                            PASSED  [47%]
test_only_asset_listing_or_pool_admins                       PASSED  [47%]
test_only_risk_or_pool_admins                                PASSED  [48%]
test_supply                                                  PASSED  [49%]
test_withdraw                                                PASSED  [50%]
test_withdraw_bad_hf                                         PASSED  [50%]
test_finalize_transfer                                       PASSED  [51%]
test_finalize_transfer_bad_hf                                PASSED  [52%]
test_set_user_use_reserve_as_collateral                      PASSED  [52%]
test_set_user_use_reserve_as_collateral_bad_hf               PASSED  [53%]
test_set_user_use_reserve_as_collateral_twice                PASSED  [54%]
test_set_user_use_reserve_as_collateral_isolation_mode       PASSED  [54%]
test_validate_supply_paused                                  PASSED  [55%]
test_validate_supply_frozen                                  PASSED  [56%]
test_validate_supply_active                                  PASSED  [56%]
test_validate_supply_amount_zero                             PASSED  [57%]
test_validate_supply_cap                                     PASSED  [58%]
test_validate_withdraw_paused                                PASSED  [59%]
test_validate_withdraw_frozen                                PASSED  [59%]
test_validate_withdraw_amount_zero                           PASSED  [60%]
test_validate_withdraw_insufficient_balance                  PASSED  [61%]
test_validate_borrow_paused                                  PASSED  [61%]
test_validate_borrow_frozen                                  PASSED  [62%]
test_validate_borrow_amount_zero                             PASSED  [63%]
test_validate_borrow_borrwoing_disabled                      PASSED  [63%]
test_validate_borrow_price_oracle_sentinel_disallowed        PASSED  [64%]
test_validate_borrow_interest_rate_mode                      PASSED  [65%]
test_validate_borrow_borrow_cap                              PASSED  [65%]
test_validate_borrow_isolation_not_borrowable                PASSED  [66%]
test_validate_borrow_isolation_debt_ceiling                  PASSED  [67%]
test_validate_borrow_emode_category                          PASSED  [68%]
test_validate_borrow_zero_collateral                         PASSED  [68%]
test_validate_borrow_bad_hf                                  PASSED  [69%]
test_validate_borrow_insufficient_collateral                 PASSED  [70%]
test_validate_borrow_stable_disabled                         PASSED  [70%]
test_validate_borrow_collateral_currency                     PASSED  [71%]
test_validate_borrow_stable_max_size                         PASSED  [72%]
test_validate_repay_paused                                   PASSED  [72%]
test_validate_repay_amount_zero                              PASSED  [73%]
test_validate_repay_same_block_stable                        PASSED  [74%]
test_validate_repay_same_block_variable                      PASSED  [75%]
test_validate_repay_no_debt                                  PASSED  [75%]
test_validate_repay_max_on_behalf_of                         PASSED  [76%]
test_validate_swap_rate_mode_paused                          PASSED  [77%]
test_validate_swap_rate_mode_frozen                          PASSED  [77%]
test_validate_swap_rate_mode_no_stable_debt                  PASSED  [78%]
test_validate_swap_rate_mode_no_variable_debt                PASSED  [79%]
test_validate_swap_rate_mode_stable_disabled                 PASSED  [79%]
test_validate_swap_rate_mode_stable_collateral               PASSED  [80%]
test_validate_rebalance_stable_borrow_rate_paused            PASSED  [81%]
test_validate_rebalance_stable_borrow_rate_criteria          PASSED  [81%]
```

```
test_validate_set_use_reserve_as_collateral_paused       PASSED  [82%]
test_validate_set_use_reserve_as_collateral_no_balance   PASSED  [83%]
test_validate_flash_loan_paused                          PASSED  [84%]
test_validate_flash_loan_simple_paused                   PASSED  [84%]
test_validate_liquidation_call_paused                    PASSED  [85%]
test_validate_liquidation_call_sentinel                  PASSED  [86%]
test_validate_liquidation_call_health_factor             PASSED  [86%]
test_validate_liquidation_call_collateral_not_enabled    PASSED  [87%]
test_validate_liquidation_call_no_debt                   PASSED  [88%]
test_validate_health_factor                              PASSED  [88%]
test_validate_hf_and_ltv                                 PASSED  [89%]
test_validate_transfer                                   PASSED  [90%]
test_validate_drop_reserve_zero_address                  PASSED  [90%]
test_validate_drop_reserve_invalid_asset                 PASSED  [91%]
test_validate_drop_reserve_atoken                        PASSED  [92%]
test_validate_drop_reserve_stable                        PASSED  [93%]
test_validate_drop_reserve_variable                      PASSED  [93%]
test_validate_set_user_emode_invalid_category            PASSED  [94%]
test_validate_set_user_emode_category_match_borrowings   PASSED  [95%]
test_getters                                             PASSED  [95%]
test_wad_mul                                             PASSED  [96%]
test_wad_div                                             PASSED  [97%]
test_ray_mul                                             PASSED  [97%]
test_ray_div                                             PASSED  [98%]
test_ray_to_wad                                          PASSED  [99%]
test_ray_mul_div_rounding                                PASSED  [100%]
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

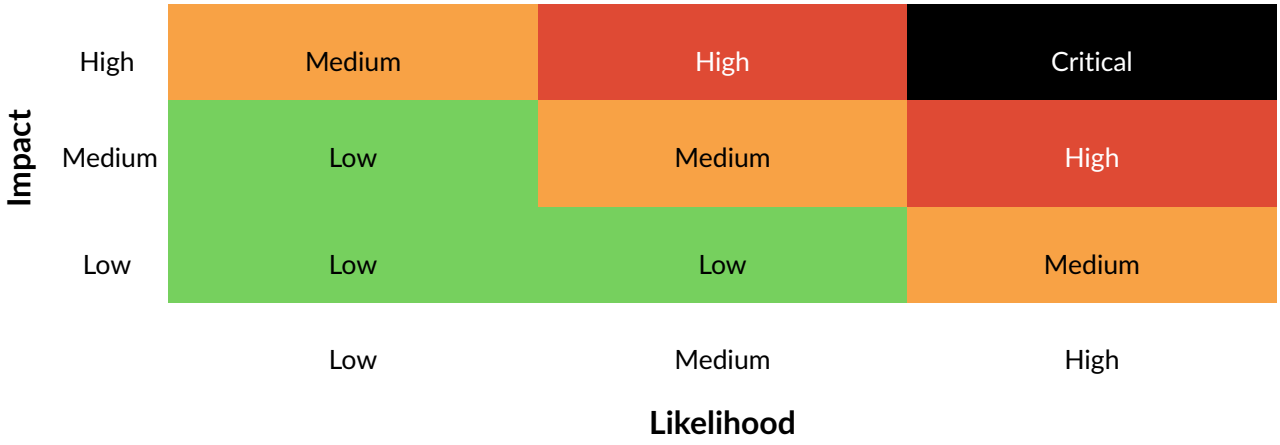

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.