AAVE PROTOCOL V2 SMART CONTRACTS AUDIT

December 3, 2020

AAVE

MixBytes()

TABLE OF CONTENTS

01 INTRODUCTION TO THE AUDIT

General Provisions

Aave is a decentralized non-custodial money market protocol where users can participate as depositors or borrowers. Depositors provide liquidity to the market to earn a passive income, while borrowers are able to borrow in an overcollateralized (perpetually) or undercollateralized (one-block liquidity) fashion.

Scope of audit

LendingPoolAddressesProvider.sol
LendingPoolAddressesProviderRegistry.sol
IAaveIncentivesController.sol
IChainlinkAggregator.sol
IERC20.sol
IERC20Detailed.sol
IExchangeAdapter.sol
ILendingPool.sol
ILendingPoolAddressesProvider.sol
ILendingPoolAddressesProviderRegistry.sol
ILendingRateOracle.sol
IPriceOracle.sol
IPriceOracleGetter.sol
IReserveInterestRateStrategy.sol
ISwapAdapter.sol
IUniswapExchange.sol
DefaultReserveInterestRateStrategy.sol
LendingPool.sol
LendingPoolCollateralManager.sol
LendingPoolConfigurator.sol
LendingPoolStorage.sol
ReserveConfiguration.sol
UserConfiguration.sol
GenericLogic.sol
ReserveLogic.sol
ValidationLogic.sol
Errors.sol
Helpers.sol
GenericLogic.sol
ReserveLogic.sol
ValidationLogic.sol
MathUtils.sol
PercentageMath.sol
Context.sol
IERC20DetailedBytes.sol
AToken.sol
IncentivizedERC20.sol
StableDebtToken.sol
VariableDebtToken.sol
DebtTokenBase.sol
IAToken.sol
IScaledBalanceToken.sol
IStableDebtToken.sol
IVariableDebtToken.sol

02 SECURITY ASSESSMENT PRINCIPLES

Classification of Issues

- CRITICAL: Bugs leading to Ether or token theft, fund access locking or any other loss of Ether/tokens to be transferred to any party (for example, dividends).

- MAJOR: Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.

- WARNINGS: Bugs that can break the intended contract logic or expose it to DoS attacks.

- COMMENTS: Other issues and recommendations reported to/ acknowledged by the team.

Security Assessment Methodology

Two auditors independently verified the code.

Stages of the audit were as follows:

- “Blind” manual check of the code and its model
- “Guided” manual code review
- Checking the code compliance with the customer requirements
- Discussion of independent audit results
- Report preparation

03 DETECTED ISSUES

CRITICAL

Not found

MAJOR

1. ReserveConfiguration.sol#L18

One leading `F` is missing in this mask.

```
uint256 constant LIQUIDATION_BONUS_MASK = 0xFFFFFFF0000FFFFFFFF;
```

This fact results in the corruption of four bits in the reserve factor field during the operations on the liquidation bonus field. `getLiquidationBonus` is also affected.
We suggest fixing the mask. We also recommend adding tests for each `ReserveConfiguration` field which have other fields set to some values, perform some actions, restore the field to the original value and make sure that the entire mask is intact.

Status

Fixed at *5ddd98c5*

2. ReserveConfiguration.sol#L46

ReserveConfiguration.sol#L63

ReserveConfiguration.sol#L84

ReserveConfiguration.sol#L106

ReserveConfiguration.sol#L128

Bit lengths of the provided values are not checked against bit lengths of the corresponding fields in the `data`.

For example, here

```
function setLtv(ReserveConfiguration.Map memory self, uint256 ltv) internal
pure {
    self.data = (self.data & LTV_MASK) | ltv;
```

providing a value greater than `65535` as `ltv` will result in a corruption of the liquidation threshold field. We recommend making sure that passed values fit in the corresponding fields.

Status

Fixed at ***b3cc9d1a***

3. ValidationLogic.sol#L68

It looks like `userBalance` is erroneously used instead of `amount`. This will result in overly strict restrictions on withdrawals. We suggest replacing the argument.

Status

Fixed at ***b4f85927***

4. ValidationLogic.sol#L200

The `vars.availableLiquidity` field is not initialized before usage.

```
//calculate the max available loan size in stable rate mode as a percentage of
the
//available liquidity
uint256 maxLoanSizeStable =
vars.availableLiquidity.percentMul(maxStableLoanPercent);
```

Proper initialization should be added.

Status

Fixed at ***3f714b9d***

5. LendingPoolConfigurator.sol#L533

Changing the decimals here will not automatically change the decimals either in the aToken or in the debt tokens.

```
function setReserveDecimals(address asset, uint256 decimals) external
onlyAaveAdmin {
    ReserveConfiguration.Map memory currentConfig =
pool.getConfiguration(asset);

    currentConfig.setDecimals(decimals);

    pool.setConfiguration(asset, currentConfig.data);
```

Additionally, the oracle must be updated simultaneously to consider the new value of the decimals. Otherwise, significant mispricing and liquidations may occur. We suggest removing this function. Alternatively, the change may be allowed only for inactive reserve and must be propagated to the tokens.

Status

Fixed at ***bfa26634***

6. ReserveLogic.sol#L341-347

Value `vars.previousStableDebt` calculated this way is actually the current stable debt and always equals to `vars.currentStableDebt`.

```
//calculate the stable debt until the last timestamp update
vars.cumulatedStableInterest = MathUtils.calculateCompoundedInterest(
  vars.avgStableRate,
  vars.stableSupplyUpdatedTimestamp
);

vars.previousStableDebt =
vars.principalStableDebt.rayMul(vars.cumulatedStableInterest);
```

As a result, the stable debt difference is not taken into account. Moreover, the processed stable debt increment is not recorded in any way.

One possible solution is to treat `vars.principalStableDebt` as the previous stable debt and update `StableDebtToken`'s `_totalSupply` and `_totalSupplyTimestamp` after the operation.

Status

Fixed at ***276dee49***

7. LendingPool.sol#L582

```
IERC20(asset).safeTransferFrom(receiverAddress, vars.aTokenAddress,
vars.amountPlusPremium);

reserve.updateState();
reserve.cumulateToLiquidityIndex(IERC20(vars.aTokenAddress).totalSupply(),
vars.premium);
reserve.updateInterestRates(asset, vars.aTokenAddress, vars.premium, 0);
```

LendingPoolCollateralManager.sol#L521

```
IERC20(toAsset).safeTransferFrom(
    receiverAddress,
    address(vars.toReserveAToken),
    vars.amountToReceive
);

if (vars.toReserveAToken.balanceOf(msg.sender) == 0) {
    _usersConfig[msg.sender].setUsingAsCollateral(toReserve.id, true);
}

vars.toReserveAToken.mint(msg.sender, vars.amountToReceive,
toReserve.liquidityIndex);
toReserve.updateInterestRates(
    toAsset,
    address(vars.toReserveAToken),
    vars.amountToReceive,
    0
);
```

`updateInterestRates` needs to be called with `liquidityAdded` set to `0` since liquidity was already transferred to the pool's balance. Otherwise, overestimated liquidity would lead to too low debt interest rates.

Status

Fixed at a2e2450b

8. LendingPoolCollateralManager.sol#L227-232

```
principalReserve.updateInterestRates(
  principal,
  principalReserve.aTokenAddress,
  vars.actualAmountToLiquidate,
  0
);

if (vars.userVariableDebt >= vars.actualAmountToLiquidate) {
  IVariableDebtToken(principalReserve.variableDebtTokenAddress).burn(
    user,
    vars.actualAmountToLiquidate,
    principalReserve.variableBorrowIndex
  );
} else {
  IVariableDebtToken(principalReserve.variableDebtTokenAddress).burn(
    user,
    vars.userVariableDebt,
    principalReserve.variableBorrowIndex
  );

  IStableDebtToken(principalReserve.stableDebtTokenAddress).burn(
    user,
    vars.actualAmountToLiquidate.sub(vars.userVariableDebt)
  );
}
```

LendingPoolCollateralManager.sol#L409-414

```
debtReserve.updateInterestRates(
  principal,
  vars.principalAToken,
  vars.actualAmountToLiquidate,
  0
);
IERC20(principal).safeTransferFrom(receiver, vars.principalAToken,
vars.actualAmountToLiquidate);

if (vars.userVariableDebt >= vars.actualAmountToLiquidate) {
  IVariableDebtToken(debtReserve.variableDebtTokenAddress).burn(
    user,
    vars.actualAmountToLiquidate,
    debtReserve.variableBorrowIndex
  );
} else {
  IVariableDebtToken(debtReserve.variableDebtTokenAddress).burn(
    user,
    vars.userVariableDebt,
    debtReserve.variableBorrowIndex
  );
  IStableDebtToken(debtReserve.stableDebtTokenAddress).burn(
    user,
    vars.actualAmountToLiquidate.sub(vars.userVariableDebt)
  );
}
```

Debt reserve interest rates are updated before debt burning takes place.

As a result, stale total debt values are used during interest rates calculation. We suggest switching `updateInterestRates` and the `if` statement.

Status

Fixed at c5d7bb5e

9. LendingPool.sol#L511

The `liquidityAdded` parameter of the `updateInterestRates` call seems to be incorrect as the flashloan body is yet to be transferred thus it will not be included in the interest rates calculation.

Status

Fixed at 584a5676

WARNINGS

1. LendingPoolAddressesProviderRegistry.sol#L38-50

Unoptimized usage of storage-allocated list `addressesProvidersList`.

Reading of a single element of a list requires 2 `SLOAD`s (due to overflow checks), and loop bounds check requires 1 `SLOAD` (3*N `SLOAD`s total).

Caching `uint[] memory _addressesProvidersList = addressesProvidersList;` requires only N+1 `SLOAD`s.

We recommend to rewrite the function as

```
function getAddressesProvidersList() external override view returns (address[]
memory) {
    uint256 maxLength = addressesProvidersList.length;

    address[] memory _addressesProvidersList = addressesProvidersList;
    address[] memory activeProviders = new address[]
(_addressesProvidersList.length);

    for (uint256 i = 0; i < _addressesProvidersList.length; i++) {
    if (addressesProviders[_addressesProvidersList[i]] > 0) {
        activeProviders[i] = addressesProvidersList[i];
      }
    }

    return activeProviders;
}
```

Status

Fixed at ***8a82c8f1***

2. LendingPoolAddressesProviderRegistry.sol#L77-81

```
for (uint256 i = 0; i < addressesProvidersList.length; i++) {
  if (addressesProvidersList[i] == provider) {
    return;
  }
}
```

Unoptimized loop. `addressesProvidersList.length` provides multiple `SLOAD`s inside the loop. We recommend you to replace a storage allocated variable to a memory cached one.

Status

Fixed at ***c14ea749***

3. LendingPool.sol#L953

```
for (uint256 i = 0; i < _reservesList.length; i++)
if (_reservesList[i] == asset) {
    reserveAlreadyAdded = true;
}
if (!reserveAlreadyAdded) {
  _reserves[asset].id = uint8(_reservesList.length);
  _reservesList.push(asset);
}
```

Unoptimized `reserveAlreadyAdded` computation. It could be computed with the following expression with O(1) complexity:

```
bool reserveAlreadyAdded = _reserves[asset].id != 0 || (_reservesList.length >
0 &&_reservesList[0]==asset);
```

Status

Fixed at *232743c3*

4. MathUtils.sol#L27-30

We recommend replacing computation with the following. Precision is better, gas cost is smaller. Also, the same types are used for time, as in `calculateCompoundedInterest`.

```
return rate.mul(timeDifference).div(SECONDS_PER_YEAR).add(WadRayMath.ray());
```

Status

Fixed at *e88d9dc8*

5. ReserveConfiguration.sol#L23

The mask is incorrect for a bit field manipulation.

```
uint256 constant STABLE_BORROWING_MASK = 0xFFFF07FFFFFFFFFFFFFF;
```

It looks like it should be `0xFFFFF7FFFFFFFFFFFFFF`. However, the issue doesn't cause troubles at the moment since bits 60-63 are presently unused. If they are unused on purpose, we suggest explicitly stating it in the Map's comment.

Status

Fixed at *5ddd98c5*

6. IncentivizedERC20.sol#L83

DebtTokenBase.sol#L29

`msg.sender` is erroneously used instead of `_msgSender()`. Since the contracts inherit from `Context` they should use `_msgSender()` instead of `msg.sender` to properly support `Context`.

The issue comes up when some GSN-like solutions are used together with the tokens. The issue affects off-chain clients and DApps listening for the event. We suggest making an appropriate replacement. Also, we recommend getting rid of `msg.sender` mentions in the files altogether.

Status

Fixed at 87bbfb95

7. LendingPool.sol#L919

Checking `reservesCount > 0` is important. Otherwise `reserveAlreadyAdded` is computed wrong if `asset` is zero and `_reservesList` is empty.

Status

Acknowledged

8. ValidationLogic.sol#L113

We suggest checking that `amount != 0`.

Status

Fixed at ee1e2056

9. LendingPoolAddressesProvider.sol#L46-49

It is not tracked if a proxy is used for each particular address id. As a result, the transition from a non-proxied mode into a proxied one is impossible (`upgradeToAndCall` will fail **LendingPoolAddressesProvider.sol#L167**). Also, an unintended transition from a proxied mode into a non-proxied one is possible. We suggest tracking the fact of proxy usage for each particular address id and making mode transition explicit.

Status

Acknowledged

10. LendingPoolAddressesProviderRegistry.sol#L57

Zero lending pool `id` is used as an indicator of an inactive/absent lending pool. Therefore, we suggest prohibiting passing `0` as an `id` parameter value.

```
function registerAddressesProvider(address provider, uint256 id) external
override onlyOwner {
  _addressesProviders[provider] = id;
```

Status

Fixed at b3cc9d1a

11. AToken.sol#L45

`msg.sender` is erroneously used instead of `_msgSender()`. Since the contracts inherit from `Context` they should use `_msgSender()` instead of `msg.sender` to properly support `Context`. We suggest making an appropriate replacement.

Status

Fixed at b3cc9d1a

12. AToken.sol#L111

`msg.sender` is erroneously used instead of `user`. We suggest making an appropriate replacement.

Status

Fixed at 9fddcd0a

13. IncentivizedERC20.sol#L126-130

During this call, an `Approval` event will be emitted. It is not obvious that an allowance decrease during a `transferFrom` operation should fire this event. Event listeners could mix this up with “regular“ `Approval` events happening during `approve` and similar operations.

Status

Acknowledged

14. AToken.sol#L153-155

A `Transfer` event is not emitted during `_transfer` and `super._transfer`. As a result, event listeners will not be able to track transfers of collateral during liquidations. We recommend emitting an appropriate event.

Status

Fixed at *727bc12d*

15. StableDebtToken.sol#L121-124

An average stable rate calculated this way will not compound equivalently to a pair of individual debt positions as the time passes. The cause is the exponentiation during the debt calculation. A simple script illustrates the issue:

```
SECONDS_PER_YEAR = 365 * 86400
delta_t = 2 * SECONDS_PER_YEAR

balanceOf = lambda principal, rate: principal * ((1. + rate / SECONDS_PER_YEAR)
** delta_t)

principal_A = 200
rate_A = 0.5
principal_B = 500
rate_B = 0.1

principal_sum = principal_A + principal_B
rate_avg = (principal_A * rate_A + principal_B * rate_B) / principal_sum

debt_separate = balanceOf(principal_A, rate_A) + balanceOf(principal_B, rate_B)
debt_unified = balanceOf(principal_sum, rate_avg)

print('Separate accounts total debt: {:.0f}'.format(debt_separate))
print('Avg rate of unified debt: {:.2f}'.format(rate_avg))
print('Total unified debt: {:.0f}'.format(debt_unified))
print('Debt calculation difference: {:.0f}%'.format(100 * (debt_separate -
debt_unified) / debt_separate))
```

Which yields:

```
Separate accounts total debt: 1154
Avg rate of unified debt: 0.21
Total unified debt: 1075
Debt calculation difference: 7%
```

Make sure that this behavior is acceptable.

Status

Acknowledged. See below.

16. StableDebtToken.sol#L134-137

StableDebtToken.sol#L178-181

Due to the reason stated in the previous issue, it is incorrect to calculate `totalSupply` based on the average stable rate calculated this way. An important consequence is that `totalSupply` will not match the sum of all user debt balances. Moreover, as the time passes it may significantly drift away from the right value. At the moment it is impossible to accurately know on-chain the total stable debt of an asset. Underestimated debt, in its turn, affects the interest rates. Make sure that this risk is acceptable.

Status

Acknowledged

Client: yes, not a problem. It's part of the issue that it's impossible to calculate an accurate avgStableRate onchain, because of the compounding on the interest. It will create an excess of interest generated by the borrowers that is not distributed to depositors (most likely to be handled when this version of the protocol will be dismissed). It will be up to the governance when we are in this situation to decide what to do with the excess.

17. StableDebtToken.sol#L142

Accrued interest added to the principal (`balanceIncrease`) is not added to the `amount` during the `Transfer` event emission.

```
_mint(user, amount.add(balanceIncrease), vars.previousSupply);

// transfer event to track balances
emit Transfer(address(0), user, amount);
```

As a result, event listeners will not be able to track the stable debt of the user without consulting the `balanceOf` function. Make sure that this is the desired behavior.

Status

Acknowledged

The event is implemented this way for the sake of consistency between the debt and liquidity tokens.

18. StableDebtToken.sol#L202

Accrued interest (`balanceIncrease`) is not taken into consideration during the `Transfer` event emission.

```
if (balanceIncrease > amount) {
  _mint(user, balanceIncrease.sub(amount), previousSupply);
} else {
  _burn(user, amount.sub(balanceIncrease), previousSupply);
}

// transfer event to track balances
emit Transfer(user, address(0), amount);
```

As a result, event listeners will not be able to track the stable debt of the user without consulting the `balanceOf` function. In some cases the stable debt decreases and in some cases it increases. We suggest emitting the events in the branches of the `if` statement with the proper values.

Status

Acknowledged

The event is implemented this way for the sake of consistency between the debt and liquidity tokens.

19. LendingPoolConfigurator.sol#L440

A reserve increase as a result of a `swapLiquidity` operation is enabled for a reserve in the frozen state. Make sure that this is the desired behavior.

Status

Fixed at b7efa920

20. ValidationLogic.sol#L59

LendingPool.sol#L366

LendingPool.sol#L554

There are no checks that the reserve is active. As a result, `withdraw`, `rebalanceStableBorrowRate`, `flashLoan` operations are enabled for an inactive reserve. Make sure that this is the desired behavior.

Status

Fixed at 57ed9efd, f87873a6

21. LendingPool.sol#L158

In the case of a bank run on a reserve, there is a period when the reserve can not service all withdrawal requests. This liquidity deficit will last until high interest rates (caused by the high usage ratio) kick in. Potentially it may take a long time.

In an adverse edge case of the fast devaluation of an asset, debtors may prefer to keep the debt despite high interest rates, profiting from the short position. In this case, lenders are stuck in lossmaking long positions. One possible solution is to use exponential interest rates in the 98% - 100% usage ratio scenario instead of a linear slope in `DefaultReserveInterestRateStrategy`.

Status

Acknowledged

Client: no action, consequence on the model and managed on the interest strategies.

22. GenericLogic.sol#L250

During a health factor calculation, liquidation bonuses must be taken into account one way or another. Otherwise, the user's collateral would not have enough funds to repay the borrowed assets and liquidator's bonuses. The first possible solution is to include bonuses explicitly in the HF calculation. Another solution is to ensure that an asset liquidation threshold is lower than `1 / liquidationBonus` (although some extra margin should be included to tackle the price slippage).

Status

Fixed at *43d64c45*

23. ValidationLogic.sol#L180-181

ValidationLogic.sol#L194

ValidationLogic.sol#L278-279

These checks can be entirely bypassed as it is not enforced during deposits or other increases of the user's aToken balance.

Consider this sequence of operations:

- User deposits asset B
- User borrows asset A
- User deposits asset A
- User withdraws asset B

Eventually, the user will be able to have any amount of deposited funds in the same asset as long as it is bigger than the debt.

Moreover, since interest rate strategies do not work on a per-user basis, an attacker can employ multiple accounts to manipulate reserve rates and bypass these checks.

Status

Acknowledged

Client: if you bypass the condition, you will end up with a higher stable rate.

24. LendingPoolCollateralManager.sol#L253-275

There is no `userConfig.setUsingAsCollateral(collateralReserve.id, false)` call in the case of the total depletion of this kind of user's collateral.
We suggest adding the call.

Status

Fixed at a3ee5d2c

25. LendingPoolCollateralManager.sol#L301

This requirement is not satisfied - the collateral may be completely liquidated, as it can be seen below:

```
vars.maxPrincipalAmountToLiquidate =
vars.userStableDebt.add(vars.userVariableDebt);
```

Status

Fixed at b7efa920

26. LendingPoolCollateralManager.sol#L415

There may be some amount of the principal left behind on the `receiver`'s balance because of a stale or time-averaged conversion price provided by the oracle or because of the liquidation bonus paid.
Also, if a position is liquidated by the owner, the owner unnecessary pays the liquidation bonus. Make sure that this is the desired behavior.

Status

Fixed at ***b7efa920***

27. LendingPoolCollateralManager.sol#L309

A liquidator does not get any bonus for calling `repayWithCollateral`. Make sure that this is the desired behavior.

Status

Fixed at ***b7efa920***

28. LendingPoolCollateralManager.sol#L505

There may be some dust of the `toAsset` on the balance of `receiverAddress` sent by a third party or funds of other users. `LendingPoolCollateralManager` would not have the approval to spend these funds, rendering any `swapLiquidity` operations via the `receiverAddress` impossible.

Status

Fixed at ***b7efa920***

29. VersionedInitializable.sol#L19-27

We suggest using explicitly computed storage slots for any fields responsible for proxy mechanics since their location must be consistent across the code versions and any collisions with business fields must be prevented. A good example of the explicitly computing storage slots technique can be found **BaseUpgradeabilityProxy.sol#L20-37**.

As a possible issue example consider **DebtTokenBase.sol#L20**. The field order is determined by the inheritance order, `VersionedInitializable` fields go after `IncentivizedERC20`. Any addition of new non-mapping fields in a new version of `IncentivizedERC20` would overwrite the `lastInitializedRevision` and `initializing` fields.

The issue proof of concept:
https://github.com/Eenae/VersionedInitializable-issue-PoC

Status

Acknowledged

Client: we will make sure the future implementations keep track of the proper chain of inheritance and storatge layout.

30. LendingPoolConfigurator.sol#L487

LendingPoolConfigurator.sol#L503

LendingPoolConfigurator.sol#L518

LendingPoolConfigurator.sol#L344-345

We suggest specifying units of measurement for these parameters as well as acceptable value ranges in the comments. Parameter validation will be helpful as well. Incorrectly set `ltv` and `liquidationThreshold` lead to fund losses.

Status

Fixed at 92e2ecab

31. LendingPool.sol

`_reservesList` is fully loaded to memory on a huge set of user's actions. If `SLOAD` cost is increased up to 2100 at EIP-2929, for 128-sized list length it will take 2100*(128+1)=270900 gas.

In most cases, it is enough to get information only about the reserve touched by the user at the transaction.

Status

Fixed at 7a0d201f

32. LendingPoolConfigurator.sol#L506

When setting `threshold` to zero, some positions may instantly become undercollaterized and, what is more important, non-liquidatable. We suggest requiring `availableLiquidity == 0` in this case.

Another possible solution is to rewrite **ValidationLogic.sol#L362-363**

```
    bool isCollateralEnabled =
collateralReserve.configuration.getLiquidationThreshold() > 0 &&
      userConfig.isUsingAsCollateral(collateralReserve.id);
```

as

```
    bool isCollateralEnabled =
userConfig.isUsingAsCollateral(collateralReserve.id);
```

so that collateral could be liquidated if and only if the user allows it so. Plus, isUsingAsCollateral should be set to true by default only if `getLiquidationThreshold() > 0` and `setUserUseReserveAsCollateral(true)` should be allowed only if `getLiquidationThreshold() > 0`. Liquidation of collateral by itself won't be a problem if the user authorized it and if the oracle provides a fair price for the asset. That way you will be able to instantly switch off some collateral, a lot of positions may become liquidatable, but liquidations will be fair (except for some losses on liquidation bonuses). Graceful threshold decreases and public warnings are still recommended.

Status

***Fixed at** 948bd960*

33. LendingPoolCollateralManager.sol#L255

There is no `_usersConfig[msg.sender].setUsingAsCollateral(collateralReserve.id, true)` call in the case when the liquidator received an amount of collateral aToken for the first time. We suggest adding the call.

Status

***Fixed at** 3f070d67*

34. LendingPoolCollateralManager.sol#L415-433

There is no `userConfig.setBorrowing(debtReserve.id, false);` call in the case of complete liquidation of the `user` debt. We suggest adding the call.

Status

***Fixed at** b7efa920*

35. LendingPool.sol#L593

```
//if the user didn't choose to return the funds, the system checks if there
//is enough collateral and eventually open a position
 _executeBorrow(
   ExecuteBorrowParams(
     asset,
     msg.sender,
     msg.sender,
     vars.amountPlusPremium,
     mode,
     vars.aTokenAddress,
     referralCode,
     false
   )
 );
```

`vars.amountPlusPremium` is used as a loan principal instead of `amount`. It means that eventually the pool will be repaid extra `vars.premium` above the lender deposits. These extra funds are not accounted and distributed either to the treasury or to lenders. An extreme example can be described: suppose someone pulls the entire asset as a flashloan with the variable rate mode and then repays it in the same block. The procedure is repeated 100 times. At the end of the block the pool asset balance is 9% more than before the block. The liquidity and variable borrow indexes did not change since the beginning of the block, so the flashloans did not accrue any interest. The 9% will remain undistributed.

We suggest either using `amount` as a flashloan borrow principal or distributing the premium using `cumulateToLiquidityIndex` or eliminating flashloan borrows at all.

Status

Fixed at a2e2450b

36. Interest excess in case of a disabled stable rate

In the case of disabled stable rate borrowing, there will be some excess interest for the asset because borrowers pay a compounded interest and lenders receive a linear one. That interest won't be withdrawable in any way (only a code upgrade will help). Make sure that this is the desired behavior.

Status

Acknowledged.

Client: it is intended, V1 is like that as well. The idea was that this would create a small cushion between what the borrowers are paying and what the depositors are receiving, so to account for potential rounding errors (at least it should ensure that there isn't liquidity missing for depositors withdrawal at the end, even if this means getting slightly underpaid). Conceptually I don't think the difference will be much higher, for two reasons: 1. The compounding on borrowing is approximated so the borrowers are already slightly undercharged. 2. The interest on the depositors' side is linear between actions, but it compounds on every action so with an activity like the one we are seeing in V1, the difference compared to a pure compounded interest is most likely negligible with the positive side effect of being less gas-intensive.

37. High stable borrowing rate rebalance threshold

If the utilization rate of an asset is below 95%, no rebalances happen. Suppose a lot of stable debt was borrowed cheaply, and then the liquidity rate goes over the stable borrowing rate of some position due to a market move. That would endanger the solvency of the asset pool.

Status

Acknowledged.

Client conveyed that the ultimate solution in this rare case is a change of `REBALANCE_UP_USAGE_RATIO_THRESHOLD` *via a code upgrade.*

38. GenericLogic.sol#L99-109

In all appearances, Loan To Value and Liquidation Threshold are initial margin and maintenance margin in classic terms. A user cannot borrow further if the weight-averaged Loan To Value is reached. However, they can withdraw collateral as long as the position is above the Liquidation Threshold. This way using transient collateral, obtained with a flashloan, perhaps, users can bypass the Loan To Value check rendering it ineffective. We suggest using Loan To Value while deciding on a balance decrease approval.

Status

Acknowledged

Client: we had this discussion internally for quite a long time, and we were aware of the fact that LTV was a soft restriction and could be bypassed. The goal of the LTV is mainly to avoid having normal users opening a position and getting instantly liquidated, which would happen if we use the ltv for both borrowing power and maintenance margin.

39. ReserveLogic.sol#L253

Real token balance could be manipulated by flashloans. We recommend to use virtual balances (not affected by flashloans) for available liquidity computations.

Status

Acknowledged

40. LendingPoolConfigurator.sol#L376

The constraint seems to be incorrect as `liquidationBonus` is in inverse relation to `liquidationThreshold` - the lower the threshold the more excess collateral is available as a bonus.

```
//we also need to require that the liq threshold is lower or equal than the
liquidation bonus, to ensure that
//there is always enough margin for liquidators to receive the bonus.
require(liquidationThreshold.add(absoluteBonus) <=
PercentageMath.PERCENTAGE_FACTOR, Errors.LPC_INVALID_CONFIGURATION);
```

Consider `liquidationThreshold = 1%` as an extreme example. The current constraint limits `liquidationBonus` to `199%` in this case. However, since liquidation happens when collateral * 0.01 < debt, the market value of the collateral can be up to one hundred times the value of the debt i.e., `liquidationBonus` can be up to 10 000%.

In fact, `liquidationThreshold * liquidationBonus` should be less or equal to `PERCENTAGE_FACTOR`.

Status

Fixed at 43d64c45

41. ValidationLogic.sol#L194

The current stable debt of the user is not added to the `amount` in this comparison.

```
require(
    !userConfig.isUsingAsCollateral(reserve.id) ||
      reserve.configuration.getLtv() == 0 ||
      amount > IERC20(reserve.aTokenAddress).balanceOf(userAddress),
    Errors.CALLATERAL_SAME_AS_BORROWING_CURRENCY
);
```

As a result, relatively small increments to the current stable debt will be prohibited. We suggest taking the current stable debt into consideration similarly to this check **ValidationLogic.sol#L278-279** .

Status

Acknowledged

42. ValidationLogic.sol#L200

This check may be bypassed by iterative small stable debt increases. We suggest adding the total stable debt to the `amount`.

```
//calculate the max available loan size in stable rate mode as a percentage of
the
//available liquidity
uint256 maxLoanSizeStable =
vars.availableLiquidity.percentMul(maxStableLoanPercent);

require(amount <= maxLoanSizeStable,
Errors.AMOUNT_BIGGER_THAN_MAX_LOAN_SIZE_STABLE);
```

Moreover, `MAX_STABLE_RATE_BORROW_SIZE_PERCENT` constraint is not checked during `swapBorrowRateMode`, `withdraw`, `swapLiquidity` operations, as well as liquidations. That may render the constraint ineffective.

Status

Acknowledged

Client: if you bypass the condition, you will end up with a higher stable rate.

43. DelegationAwareAToken.sol#L55-63

A new `initialize` implementation shadows the base contract implementation. As a result, the `DOMAIN_SEPARATOR` field is uninitialized, resulting in broken `permit` functionality. We suggest removing the new implementation.

Status

Fixed at b2a871f8

44. Reentrancy in the liquidation process

The difficulty of the exploitation is high (see prerequisites), however, the impact is high as well.

Prerequisites:
• Token T is used as an asset with a non-zero liquidation threshold and loan to value.
• Token T issues a callback to the `receiver` on `transfer(receiver, amount)`.

Attack scenario:
1. An attacker has 100 T tokens as collateral.
2. Health factor of the attacker's position goes below 1 (position becomes liquidatable).
3. The attacker liquidates himself, selecting T as collateral to liquidate, opting not to `receiveAToken`, and providing such amount of `debtToCover` that `maxCollateralToLiquidate == userCollateralBalance` .
4. During the `vars.collateralAtoken.burn` call the attacker uses the reentrancy of T,
    4.1. making a 1 000 000 T deposit as the `user` .
    4.2. borrowing the maximum possible amount of other assets on behalf of the `user` .
5. The `userConfig.setUsingAsCollateral(collateralReserve.id, false);` line gets executed.
6. The attacker repays the initial debt of the liquidation.
7. Now the attacker ended up with the 1 000 000 T deposit which has the `isUsingAsCollateral` set to `false` and a huge debt.
8. The attacker now can withdraw 1 000 000 T collateral since `balanceDecreaseAllowed` returns `true` when `isUsingAsCollateral` is set to `false` .

The first remediation step is to look up the actual collateral balance here https://github.com/aave/protocol-v2/blob/750920303e33b66bc29862ea3b85206dda9ce786/contracts/protocol/lendingpool/LendingPoolCollateralManager.sol#L222. Additionally, any tokens making external calls in `transfer` and `transferFrom` should be avoided. Alternatively, a reentrancy guard can be used.

Status

Acknowledged

Client: no [token] listing can be done without deep analysis of that aspect

COMMENTS

1. ValidationLogic.sol#L53

```
function validateWithdraw(
  address reserveAddress,
  address aTokenAddress,
  uint256 amount,
  uint256 userBalance,
  mapping(address => ReserveLogic.ReserveData) storage reservesData,
  UserConfiguration.Map storage userConfig,
  address[] calldata reserves,
  address oracle
) external view {
```

Check that variable `aTokenAddress` should be unused and comment it's name.

Status

Fixed at *2e30bb8b*

2. MathUtils.sol#L11

```
uint256 internal constant SECONDS_PER_YEAR = 365 days;
```

This is not correct for leap years. We recommend you to add a comment that you ignore leap seconds or rename the variable.

Status

Fixed at *2fd3fe14*

3. ReserveConfiguration.sol#L46

```
self.data = (self.data & RESERVE_FACTOR_MASK) | reserveFactor << 64;
```

We recommend you to use the same code style for all bit mask selectors (put the right part into brackets or remove brackets to other places for the same cases).

Status

Fixed at *d56a7a27*

4. LendingPool.sol#L67

`_whenNotPaused()` is used only as a modifier. We recommend to rewrite it as a modifier.

Status

FIXED at ***3fc812e7***

5. IncentivizedERC20.sol#L184

```
uint256 totalSupply = _totalSupply;
```

We recommend avoiding global variables shadowing.

Status

Fixed at ***cb03bab6***

6. LendingPool.sol#L896

We recommend to move all bitfield-related optimizations to corresponding functions.

Here is the example for `setBorrowing`:

```
function setBorrowing(
  UserConfiguration.Map storage self,
  uint256 reserveIndex,
  bool borrowing
) internal {
  uint _data = self.data;
  uint _data_new =(_data & ~(1 << (reserveIndex * 2))) |
  (uint256(borrowing ? 1 : 0) << (reserveIndex * 2));
  if (_data != _data_new) {
      self.data = _data_new;
  }
}
```

The same optimizations could be applied to ReserveConfiguration.

Status

Fixed at ***386138cc***

7. Errors.sol

We recommend to use uint-typed codes here to reduce the size of the contract.

Status

Acknowledged

8. ReserveConfiguration.sol#L55

The mask application `& ~RESERVE_FACTOR_MASK` can be omitted because the other fields are shifted during the `>> 64` operation. We recommend removing excess code.

Status

Acknowledged

Client: no action, to not need to change the logic if we add extra fields to the mask in the future.

9. ReserveConfiguration.sol#L155

ReserveConfiguration.sol#L173

ReserveConfiguration.sol#L191

ReserveConfiguration.sol#L215

ReserveConfiguration.sol#L236-239

The bit shift operations (`>> 56`, `>> 57`, etc.) can be omitted since they do not change the boolean outcome. We recommend removing excess code.

Status

Fixed at 6cd18c43

10. LendingPoolAddressesProviderRegistry.sol#L23

The return value is documented erroneously because a number is returned, not a boolean value. We suggest correcting the comment.

Status

Fixed at a9a863fc

11. LendingPoolAddressesProviderRegistry.sol#L36

The resulting array is sparse (contains zeros for unregistered address providers), we suggest mentioning it in the documenting comment.

Status

Fixed at a9a863fc

12. LendingPoolAddressesProviderRegistry.sol#L88

The return value is documented erroneously, because in the case of absent address provider `uint256(0)` is returned. We suggest correcting the comment.

Status

Fixed at ***a9a863fc***

13. PercentageMath.sol

We recommend simplifying PercentageMath mul and div operations. Both rational mul and div reduce to computing res:=round(a * x / y), where (x, y) is the 2nd argument in rational form.

In solidity there is the following expression:

```
res = (a * x + half_y)/y;
```

We should check the term for overflow. As we can see, it is a growing function over x and half_y. It means that not overflowing `a_max(x, half_y) = floor((MAX_u256 - half_y)/x)` exists.

In solidity it could be implemented as follows:

```
require(a<=(MAX_U256 - half_y)/x);
```

This approach could be used at both WadRayMath and PercentageMath.

Status

Fixed at ***47d00a0e***

14. LendingPool.sol#L888

We recommend cache `_reservesCount`. Then there will be 3 `SLOAD` less.

Status

Partially fixed (check comment 15) at ***a9c3a033***

15. LendingPool.sol#L925

We recommend replacing `_reservesCount++;` with `_reservesCount=reservesCount+1;`, it is 1 SLOAD less.

Status

Fixed at ***ec600e56***

16. LendingPoolAddressesProvider.sol#L18

LendingPoolAddressesProvider.sol#L121

Some addresses, namely the owner of the `LendingPoolAddressesProvider` contract and the Aave admin, have substantial power over a lending pool. We suggest using multisignature or DAO solutions to control these addresses.

Status

Acknowledged

Client: it will be that way.

17. LendingPoolAddressesProvider.sol#L40

We recommend splitting this function into two separate functions since they solve different tasks which are visible in the form of two `if` statement branches.

Status

Fixed at c81047ca

18. ReserveConfiguration.sol#L88

The function is documented incorrectly. We suggest updating the comment.

Status

Fixed at 643ed2f9

19. ReserveConfiguration.sol#L227-230

ReserveConfiguration.sol#L252-255

We recommend naming the return values explicitly. That way they will be available as named properties in the JS API and in the ABI.

Status

Acknowledged

Client: no named return by our style guidelines.

20. UserConfiguration.sol#L28

UserConfiguration.sol#L44

UserConfiguration.sol#L58

UserConfiguration.sol#L72

UserConfiguration.sol#L86

Assertions can be added to ensure that `reserveIndex` does not exceed 127.

Status

Fixed at *6460dd9e*

21. LendingPoolConfigurator.sol#L29-178

Some lending pool configurator events have `asset` parameter `indexed` and some do not. Indexing allows searching for particular asset events in a block range. Make sure that no `indexed` attributes are forgotten.

Status

Fixed at *e4dc22e5*

22. LendingPoolConfigurator.sol#L373

LendingPoolConfigurator.sol#L387

Stable rate borrowing setting manipulation is permitted for a reserve with disabled borrowing. This behavior does not cause any harm, however, may be counterintuitive and even dangerous.

Status

Acknowledged

Client: no action, no risk

23. LendingPool.sol#L92

LendingPool.sol#125

LendingPool.sol#174

LendingPool.sol#189

LendingPool.sol#211

LendingPool.sol#252

LendingPool.sol#313

LendingPool.sol#365

LendingPool.sol#417

LendingPool.sol#549

LendingPool.sol#645

LendingPool.sol#680

LendingPool.sol#699

LendingPool.sol#766

LendingPool.sol#833

LendingPool.sol#841

LendingPool.sol#846

LendingPool.sol#968

LendingPool.sol#977

LendingPool.sol#993

LendingPoolCollateralManager.sol#L140-141

LendingPoolCollateralManager.sol#L310-311

LendingPoolCollateralManager.sol#L457-458

We recommend adding explicit checks that referenced assets exist. Stopping execution as soon as an error emerges is a good security practice and may prevent some complicated vulnerabilities.

Status

Fixed at ***LendingPool.sol***

24. LendingPoolConfigurator.sol#L337-339

LendingPoolConfigurator.sol#L472

LendingPoolConfigurator.sol#L503

LendingPoolConfigurator.sol#L518

We suggest checking that `ltv` is lower than `liquidationThreshold / liquidationBonus`. Otherwise, `ltv` would not serve its purpose.

Status

Fixed at *948bd960*

25. ValidationLogic.sol#L193

ValidationLogic.sol#L277

It looks like the `getLtv` function is used to check if a reserve can be used as collateral. However, `getLiquidationThreshold` is mostly used for this purpose. We recommend using a uniform approach throughout the code. Such an approach can be defined as a one expression function in `ReserveConfiguration`.

Status

Acknowledged

26. ReserveLogic.sol#L80

Actually, one unit of income has been accrued. With the original unit that gives us 2*1e27.

Status

Fixed at *fed8c798*

27. ReserveLogic.sol#L207-214

It looks like at this execution point there is no way the reserve was initialized before.

Status

Fixed at *f125eeb0*

28. LendingPool.sol#L96

LendingPool.sol#L125

LendingPool.sol#L193

LendingPool.sol#L216

LendingPool.sol#L256

LendingPool.sol#L454

LendingPool.sol#L498

LendingPool.sol#L554

LendingPool.sol#L617

We suggest using a reentrancy guard for each public-facing function that transfers any funds. This may sound like a dull and limiting step. However, it is much safer to prevent any possible reentrancy rather than to keep in mind the exponential amount of possible attack vector combinations. This article gives a prominent example.

One possible reentrancy attack vector is re-entering `deposit` and `withdraw` if the asset allows external calls before the actual token transfer. However, this reentrancy affects only `updateInterestRates` with incorrect available liquidity. Moreover, this minor incorrectness will be corrected during the next `updateInterestRates` call.

Status

Acknowledged

Client conveyed that the auditing of asset tokens, as well as the usage of the Checks-Effects-Interactions pattern, provide a sufficient level of security.

29. ValidationLogic.sol#L308-319

`balanceDecreaseAllowed` consulted even in the case when the user wants to enable the reserve as collateral.

```
require(
  GenericLogic.balanceDecreaseAllowed(
    reserveAddress,
    msg.sender,
    underlyingBalance,
    reservesData,
    userConfig,
    reserves,
    oracle
  ),
  Errors.DEPOSIT_ALREADY_IN_USE
);
```

Although, adding more collateral is not prohibited because of a check in `balanceDecreaseAllowed`. Still, we suggest calling `balanceDecreaseAllowed` only in the case of the balance decrease.

Status

Fixed at 0c8efc22

30. AToken.sol#L304

The receiver does not have the `UsingAsCollateral` flag set by default. Make sure that this is the desired behavior.

Status

Fixed at 9e55ea12

31. LendingPool.sol#L197

Borrow allowance is a subject to a double withdrawal (in this case - double borrow) attack. Plain `IERC20.approve` is subject to the same attack. More details and mitigation strategies can be found at https://blockchain-projects.readthedocs.io/multiple_withdrawal.html.

Status

Acknowledged

32. StableDebtToken.sol#L157

StableDebtToken.sol#L211

StableDebtToken.sol#L281

VariableDebtToken.sol#L30

LendingPoolConfigurator.sol#L168

LendingPoolConfigurator.sol#L176

LendingPoolConfigurator.sol#L273

LendingPoolConfigurator.sol#L286

DefaultReserveInterestRateStrategy.sol#L115-117

ValidationLogic.sol#L19

GenericLogic.sol#L146

LendingPool.sol#L441

There are some factual slips in the comments, mostly caused by copy-pasting. We suggest correcting them.

Status

Fixed at *0431f0dc*

33. ReserveLogic.sol#L169 and other 1<<128 to type(uint128).max replacings

`1<<128` is not equal to `type(uint128).max`. So, to ensure that `result` is `uint128`, non-strict checks are enough.

Status

Acknowledged

34. LendingPool.sol#L401

Typo in description for 1st parameter.

Status

Fixed at *f98335cb*

SMART CONTRACT DEPLOYMENT REVIEW

The following contracts were deployed as part of Aave protocol v2:

LendingPoolAddressesProvider

This contract is entry point of Aave protocol v2. The constructor was executed with `marketId` parameter `"Aave genesis market"`.

Bytecode is verified.

Related addresses

- Pool admin
  - 0xbd723fc4f1d737dcfc48a07fe7336766d34cad5f
  - 0xbb94a575935772d7d8ba78cd33caa64d4fb61d6b
- Emergency admin
  - 0xbd723fc4f1d737dcfc48a07fe7336766d34cad5f
- `LendingPool`
  - 0x987115c38fd9fd2aa2c6f1718451d167c13a3186
- `LendingPoolConfigurator`
  - 0x3a95ee42f080ff7289c8b4a14eb483a8644d7521
- Aave oracle:
  - 0xa50ba011c48153de246e5192c8f9258a2ba79ca9
- Lending rate oracle:
  - 0x8a32f49ffba88aba6eff96f45d8bd1d4b3f35c7d
- `LendingPoolCollateralManager`
  - 0xbd4765210d4167ce2a5b87280d9e8ee316d5ec7c

LendingPool

Bytecode is verified.

Related addresses

- `ValidationLogic`
  - 0xdd6f15b39ca5147ae9b5e6046645d55b0e5baf0c
- `ReserveLogic`
  - 0xdce33de861d200d8da88c751dc00c18eda3251f5
- `GenericLogic`
  - 0x123fba7a76b29547df94dc59933332b751206fdf

LendingPoolConfigurator

Bytecode is verified.

LendingPoolCollateralManager

Bytecode is verified.

ValidationLogic

Bytecode is verified.

ReserveLogic

Bytecode is verified.

GenericLogic

Bytecode is verified.

Summary

All contracts from the scope of the audit, deployed for Aave protocol v2, are corresponding to code freeze at https://github.com/aave/protocol-v2/tree/750920303e33b66bc29862ea3b85206dda9ce786 and https://gitlab.com/aave-tech/protocol-v2/-/tree/750920303e33b66bc29862ea3b85206dda9ce786.

04 CONCLUSION AND RESULTS

MixBytes was approached by Aave to provide a security assessment of the second version of the Aave protocol implementation. The whole audit process started on September 16 and ended on December 3, 2020. The audit effort was led by Alexey Makeev. MixBytes additionally engaged an independent highly professional contractor Igor Gulamov. The scope of the audit is listed above. Oracle, treasury, and governance implementations are out of the scope.

The protocol allows end-users to lend and borrow ERC 20 compatible tokens. Moreover, the protocol supports so-called flash loans. That, in its turn, enables several more sophisticated strategies including short selling. Key improvements introduced in the second version include debt tokenization, upgradeability of the protocol tokens, credit delegation, and gas optimizations. Lenders are rewarded with interest paid on their assets. Borrowers pay interest on their debts together with payments to the asset treasury that constitutes fund flow equilibrium for some particular assets. A particular asset in isolation, in essence, is a fractional reserve system managed by an interest rate strategy in an automated way. However, as a whole, the protocol strives to be overcollateralized. Of course, some parameters of reserves, such as liquidation thresholds, are configured externally and may require changes in accordance with the current market conditions for the protocol to stay overcollateralized.

The auditing process consisted of several stages, each of them containing several passes or checks. The scope was reviewed many times from different angles. One of the most time-consuming stages, the blind stage, took more than one month. Some local issues and slips were discovered early during this stage, e.g. bitmask manipulation issues. Crucial protocol properties were also formulated during this stage. Then, during the guided stage, the intended protocol behavior was compared with a reverse-engineered one. We rethought the protocol operations taking into consideration protocol properties. Fund flows were analyzed as well.

Subsequently, about two weeks were devoted to pattern analysis including automated checks and DeFi-specific attack vectors. Since the Aave team was providing fixes in a timely fashion, we checked the code changes shortly after providing an interim report. Although, it still took about two weeks since many checks described above were re-run to guard against newly introduced issues. That paid off a couple of times. Overall, we are impressed by the Aave team's dedication to safe code provision.

Findings list:

Level	Amount
CRITICAL	-
MAJOR	9
WARNING	43
COMMENT	34

No critical issues were found as no way to steal funds was detected. Key areas of found issues can be summarized as follows:

- Bitmask manipulation and some other misses which most likely can be attributed to the work-in-progress state of the audited code revision.

- Several `updateInterestRates` invocation issues can be described as complications of concurrent state update. There was no obvious alternative to such an approach because the Checks-Effects-Interactions pattern was used.

- User flags handling attracted our attention as it is a typical dangerous setup caused by state duplication. A few issues were discovered there.

- Some excess gas usage issues were discovered and further improvement paths were given.

- We highlighted some potential fund flow issues which ultimately have to be mitigated by interest rate strategies and/or code upgrade.

- We suggested implementing as many as possible automated checks of reserve parameters to rule out the human factor.

We see tokenization which allows self-consistent balance updates as a key security factor in the protocol even in the face of reentrancy. Simplicity and limited attack surface contribute to the protocol security as well.

The audit started when the codebase was still undergoing some changes. However, during the last weeks of the audit, the client devoted significant effort to finalize the codebase to the production-ready state. Some challenges imposed by the EVM and the Solidity compiler were faced and solved, one way or another. Some notable practices were employed, e.g. `WadRayMath` library.

We consider the commit `750920303e33b66bc29862ea3b85206dda9ce786` as a safe version from the informational security point of view.

About MixBytes

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build open-source 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://t.me/MixBytes

Disclaimer