Cod3x LEND 3 Fuzzing - Beirao audit 01/12/2024

Introduction

A time-boxed security review of the Granary V2 protocol was done by Beirao, with a focus on the security aspects of the application's smart contracts implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

About Beirao

I'm an independent smart contract security researcher. I extend my skills as a contractor specializing in EVM smart contracts security. If you're in need of robust code review, I'm here to help. We can get in touch via Twitter or Email.

About Cod3x Lend

The purpose of this audit is to highlight high level errors due to the new rehypothecation and minipool mechanism. This is not a line-by-line review, so some hidden errors may remain.

Observations

Cod3x lend is a AAVE V2 fork that adds:

- Updated pragmas (^0.8.0)
- Lending pool external rehypothecation
- The concept of minipools. Minipools are sub-markets that have a privileged borrowing capacity on the main lending pool.

Scope

The following smart contracts were in scope of the audit: (total : **8056** SLoC)

- `contracts/protocol/**`

Previous audits

Trail of Bits: Cod3x Foundation - Cod3x Lend - Comprehensive Security Assessment.pdf

Beirao: https://www.beirao.xyz/blog/SR10-Cod3x_Lend_1

Zigtur: Cod3x-Lend_Zigtur_Audit_V1.1.pdf

Beirao: https://www.beirao.xyz/blog/SR11-Cod3x_Lend_2

Severity classification

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

Impact - the technical, economic and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

***review commit hash* -** bb2a3c08

***fixes review commit hash* -** b984420c

Deployment chains

- All EVMs.

Scope

The following smart contracts were in scope of the audit: (total : **8056** SLoC)

- `contracts/protocol/**`

Findings Summary

Summary :

- **2** High(s)
- **3** Medium(s)

ID	Title	Status
H-01	`AToken6909.totalSupply()` doesn't return the correct answer if debtToken.	Fix
H-02	Rebalance rehypothecation DOS	Fix
M-01	Rounding issue in `ValidationLogic.validateWithdraw()`	Fix
M-02	Rounding issue in `ValidationLogic.validateBorrow()`	Fix
M-03	`userConfig` inconsistency	Fix

Properties

✅ : Passing
❌ : Failing

General (same for the LendingPool and MiniPools)

100. ✅ To be liquidated on a given collateral asset, the target user must own the associated `aTokenColl`.

101. ✅ To be liquidated on a given token, the target user must own the associated `vTokenDebt`.
102. ✅ `liquidationCall()` must only be callable when the target health factor is < 1.
103. ✅ `liquidationCall()` must decrease the target `vTokenDebt` balance by `amount`.
104. ✅ `liquidationCall()` must increase the liquidator `aTokenColl` (or `collAsset`) balance.
105. ✅ `liquidationCall()` must decrease the liquidator debt asset balance if `randReceiveAToken` is true or `collAsset` is n
106. ✅ `setFlowLimit()` must correctly decrease the flow.

LendingPool

200. ✅ Users must always be able to deposit in normal condition.
201. ✅ `deposit()` must increase the user aToken balance by `amount`.
202. ✅ `deposit()` must decrease the user asset balance by `amount`.
203. ✅ `withdraw()` must decrease the user aToken balance by `amount`.
204. ✅ `withdraw()` must increase the user asset balance by `amount`.
205. ✅ A user must not be able to `borrow()` if they don't own aTokens.
206. ✅ `borrow()` must only be possible if the user health factor is greater than 1.
207. ❌ `borrow()` must not result in a health factor of less than 1.
208. ✅ `borrow()` must increase the user debtToken balance by `amount`.
209. ✅ `borrow()` must decrease `borrowAllowance()` by `amount` if `user != onBehalf`.
210. ✅ `repay()` must decrease the onBehalfOf debtToken balance by `amount`.
211. ✅ `repay()` must decrease the user asset balance by `amount`.
212. ✅ `healthFactorAfter` must be greater than `healthFactorBefore` as long as liquidations are done in time.
213. ✅ `setUseReserveAsCollateral` must not reduce the health factor below 1.
214. ✅ Users must not be able to steal funds from flashloans.
215. ✅ The total value borrowed must always be less than the value of the collaterals.
216. ✅ The `liquidityIndex` should monotonically increase when there is collateral.
217. ✅ The `variableBorrowIndex` should monotonically increase when there is debt.
218. ✅ A user with debt should have at least an aToken balance `setUsingAsCollateral`.
219. ✅ Integrity of Deposit Cap - aToken supply should never exceed the cap.
220. ✅ `UserConfigurationMap` integrity: If a user has a given aToken then `isUsingAsCollateralOrBorrowing` and `isUsing
221. ✅ `UserConfigurationMap` integrity: If a user has a given debtToken then `isUsingAsCollateralOrBorrowing`, `isBorro
222. ✅ Rehypothecation: farming percentage must be respected (+/- the drift) after a rebalance occured.
223. ✅ Rehypothecation: The profit handler address must see its balance increase after reaching the claiming threshold.
224. ✅ `withdraw()` must not result in a health factor of less than 1.
225. ❌ Rehypothecation: farming percentage must be respected (+/- the drift) after any operation.

ATokens/ATokenNonRebasing

300. ✅ Zero amount transfers should not break accounting.
301. ✅ Once a user has a debt, they must not be able to transfer aTokens if this results in a health factor less than 1.
302. ✅ Transfers for more than available balance should not be allowed.
303. ✅ Transfers should update accounting correctly.
304. ✅ Self transfers should not break accounting.
305. ✅ Zero amount transfers must not break accounting.
306. ✅ Once a user has a debt, they must not be able to transfer aTokens if this results in a health factor less than 1.
307. ✅ Transfers for more than available balance must not be allowed.
308. ✅ `transferFrom()` must only transfer if the sender has enough allowance from the `from` address.
309. ✅ Transfers must update accounting correctly.
310. ✅ Self transfers must not break accounting.
311. ✅ `transferFrom()` must decrease allowance.
312. ✅ `approve()` must never revert.
313. ✅ Allowance must be modified correctly via `approve()`.
314. ✅ `increaseAllowance()` must never revert.
315. ✅ Allowance must be modified correctly via `increaseAllowance()`.

316. ✅ `decreaseAllowance()` must revert when the user tries to decrease more than currently allowed.
317. ✅ Allowance must be modified correctly via `decreaseAllowance()`.
318. ✅ Force feeding assets in LendingPool, ATokens, debtTokens, MiniPools or AToken6909 must not change the final re
319. ✅ Force feeding aToken in LendingPool, ATokens, debtTokens, MiniPools or AToken6909 must not change the final r
320. ✅ A user must not hold more than total supply.
321. ✅ Sum of users' balances must not exceed total supply.
322. ✅ `ATokenNonRebasing` `balanceOf()` should be equivalent to `ATokens` adjusted to the conversion rate.
323. ✅ `ATokenNonRebasing` `transfer()` should be equivalent to `ATokens` adjusted to the conversion rate.
324. ❌ `ATokenNonRebasing` `transferFrom()` should be equivalent to `ATokens` adjusted to the conversion rate.
325. ✅ Allowance must be modified correctly via `ATokenNonRebasing.approve()`.
326. ✅ `ATokenNonRebasing.approve()` must not modify `AToken.allowance()`.

DebtTokens

400. ✅ `approveDelegation()` must never revert.
401. ✅ Allowance must be modified correctly via `approve()`.

MiniPool

500. ✅ Users must always be able to deposit in normal condition.
501. ✅ `deposit()` must increase the user AToken6909 balance by `amount`.
502. ✅ `deposit()` must decrease the user asset balance by `amount`.
503. ✅ `withdraw()` must decrease the user AToken6909 balance by `amount`.
504. ✅ `withdraw()` must increase the user asset balance by `amount`.
505. ❌ `withdraw()` must not result in a health factor of less than 1.
506. ✅ A user must not be able to `borrow()` if they don't own AToken6909.
507. ✅ `borrow()` must only be possible if the user health factor is greater than 1.
508. ❌ `borrow()` must not result in a health factor of less than 1.
509. ✅ `borrow()` must increase the user debtToken balance by `amount` when flow borrowing is disabled.
510. ✅ `borrow()` must decrease `borrowAllowance()` by `amount` if `user != onBehalf`.
511. ✅ `repay()` must decrease the onBehalfOf debtToken balance by `amount`.
512. ✅ `repay()` must decrease the user asset balance by `amount`.
513. ✅ `healthFactorAfter` must be greater than `healthFactorBefore` as long as liquidations are done in time.
514. ✅ `setUseReserveAsCollateral` must not reduce the health factor below 1.
515. ✅ Users must not be able to steal funds from flashloans.
516. ✅ The total value borrowed must always be less than the value of the collateral when flow borrowing is disabled.
517. ✅ The `liquidityIndex` should monotonically increase when there is collateral.
518. ✅ The `variableBorrowIndex` should monotonically increase when there is debt.
519. ✅ A user with debt should have at least an AToken6909 balance `setUsingAsCollateral`.
520. ✅ Integrity of Deposit Cap - aToken supply should never exceed the cap.
521. ❌ `UserConfigurationMap` integrity: If a user has a given aToken then `isUsingAsCollateralOrBorrowing` and `isUsing
522. ✅ `UserConfigurationMap` integrity: If a user has a given debtToken then `isUsingAsCollateralOrBorrowing`, `isBorro
523. ✅ If a minipool is flow borrowing, for a given reserve, the Lendingpool liquidity interest rate remain lower than the mi
524. ✅ The aToken remainder of each assets with flow borrowing activated should remain greater than ERROR_REMAIND
525. ✅ If a minipool is flow borrowing then its address must be included in `LendingPool._minipoolFlowBorrowing`.
526. ✅ If a minipool is not flow borrowing then its address must not be included in `LendingPool._minipoolFlowBorrowing`

AToken6909

600. ✅ Zero amount transfers should not break accounting.
601. ✅ Once a user has a debt, they must not be able to transfer aTokens if this results in a health factor less than 1.
602. ✅ Transfers for more than available balance should not be allowed.
603. ✅ Transfers should update accounting correctly.
604. ✅ Self transfers should not break accounting.

605. ✅ Zero amount transfers must not break accounting.
606. ✅ Once a user has a debt, they must not be able to transfer AToken6909s if this results in a health factor less than 1.
607. ✅ Transfers for more than available balance must not be allowed.
608. ✅ `transferFrom()` must only transfer if the sender has enough allowance from the `from` address.
609. ✅ Transfers must update accounting correctly.
610. ✅ Self transfers must not break accounting.
611. ✅ `transferFrom()` must decrease allowance.
612. ✅ `approve()` must never revert.
613. ✅ Allowance must be modified correctly via `approve()`.
614. ✅ Force feeding AToken6909 in MiniPools or AToken6909 must not change the final result.
615. ✅ `approveDelegation()` must never revert.
616. ✅ Allowance must be modified correctly via `approve()`.
617. ❌ A user must not hold more than total supply.
618. ✅ Sum of users' balances must not exceed total supply.

Entry points

```
## Admin entry points

### LendingPoolAddressesProvider

- `setAddressAsProxy(bytes32 id, address implementationAddress)`
  - `setLendingPoolImpl(address pool)`
  - `setLendingPoolConfiguratorImpl(address configurator)`
- `setAddress(bytes32 id, address newAddress)`
  - `setPoolAdmin(address admin)`
  - `setPriceOracle(address priceOracle)`
  - `setMiniPoolAddressesProvider(address provider)`
  - `setFlowLimiter(address flowLimiter)`
  - `setEmergencyAdmin(address emergencyAdmin)`
  - `setPoolAdmin(address admin)`


### LendingPoolConfigurator

- `batchInitReserve(InitReserveInput[] calldata input)`
- `updateAToken(UpdateATokenInput calldata input)`
- `updateVariableDebtToken(UpdateDebtTokenInput calldata input)`
- `enableBorrowingOnReserve(address asset, bool reserveType)`
- `disableBorrowingOnReserve(address asset, bool reserveType)`
- `configureReserveAsCollateral(address asset, bool reserveType, uint256 ltv, uint256 liquidationThreshold, uint256 liquida
- `activateReserve(address asset, bool reserveType)`
- `deactivateReserve(address asset, bool reserveType)`
- `freezeReserve(address asset, bool reserveType)`
- `unfreezeReserve(address asset, bool reserveType)`
- `setCod3xReserveFactor(address asset, bool reserveType, uint256 reserveFactor)`
- `setDepositCap(address asset, bool reserveType, uint256 depositCap)`
- `setReserveInterestRateStrategyAddress(address asset, bool reserveType, address rateStrategyAddress)`
- `setPoolPause(bool val)`
- `setFarmingPct(address aTokenAddress, uint256 farmingPct)`
- `setClaimingThreshold(address aTokenAddress, uint256 claimingThreshold)`
```

- `setFarmingPctDrift(address aTokenAddress, uint256 _farmingPctDrift)`
- `setProfitHandler(address aTokenAddress, address _profitHandler)`
- `setVault(address aTokenAddress, address _vault)`
- `rebalance(address aTokenAddress)`
- `setRewarderForReserve(address asset, bool reserveType, address rewarder)`
- `setTreasury(address asset, bool reserveType, address rewarder)`
- `updateFlashloanPremiumTotal(uint128 newFlashloanPremiumTotal)`
- `enableFlashloan(address asset, bool reserveType)`
- `disableFlashloan(address asset, bool reserveType)`

MiniPoolAddressProvider

- `deployMiniPool(address miniPoolImpl, address aTokenImpl)`
- `setFlowLimit(address asset, address miniPool, uint256 limit)`
- `setMiniPoolImpl(address impl, uint256 miniPoolId)`
- `setAToken6909Impl(address impl, uint256 miniPoolId)`
- `setAddress(bytes32 id, address newAddress)`
- `setMiniPoolConfigurator(address configuratorImpl)`
- `setCod3xTreasury(uint256 id, address treasury)`

MiniPoolConfiguration

- `batchInitReserve(InitReserveInput[] calldata input, IMiniPool pool)`
- `enableBorrowingOnReserve(address asset, IMiniPool pool)`
- `disableBorrowingOnReserve(address asset, IMiniPool pool)`
- `configureReserveAsCollateral(address asset, uint256 ltv, uint256 liquidationThreshold, uint256 liquidationBonus, IMiniPo`
- `activateReserve(address asset, IMiniPool pool)`
- `deactivateReserve(address asset, IMiniPool pool)`
- `freezeReserve(address asset, IMiniPool pool)`
- `unfreezeReserve(address asset, IMiniPool pool)`
- `enableFlashloan(address asset, IMiniPool pool)`
- `disableFlashloan(address asset, IMiniPool pool)`
- `setCod3xReserveFactor(address asset, uint256 reserveFactor, IMiniPool pool)`
- `setMinipoolOwnerReserveFactor(address asset, uint256 reserveFactor, IMiniPool pool)`
- `setDepositCap(address asset, uint256 depositCap, IMiniPool pool)`
- `setReserveInterestRateStrategyAddress(address asset, address rateStrategyAddress, IMiniPool pool)`
- `setPoolPause(bool val, IMiniPool pool)`
- `setRewarderForReserve(address asset, address rewarder, IMiniPool pool)`
- `updateFlashloanPremiumTotal(uint128 newFlashloanPremiumTotal, IMiniPool pool)`

Oracle

- `setAssetSources(address[] calldata assets, address[] calldata sources, uint256[] calldata timeouts)`
- `setFallbackOracle(address fallbackOracle)`

BasePiReserveRateStrategy

- `setOptimalUtilizationRate(uint256 optimalUtilizationRate)`
- `setMinControllerError(int256 minControllerError)`
- `setPidValues(uint256 kp, uint256 ki, int256 maxITimeAmp)`

User entry points

LendingPool

- `deposit(address asset, bool reserveType, uint256 amount, address onBehalfOf)`
- `withdraw(address asset, bool reserveType, uint256 amount, address onBehalfOf)`
- `borrow(address asset, bool reserveType, uint256 amount, address onBehalfOf)`
- `repay(address asset, bool reserveType, uint256 amount, address onBehalfOf)`
- `repayWithATokens(address asset, bool reserveType, uint256 amount)`
- `setUserUseReserveAsCollateral(address asset, bool reserveType, bool useAsCollateral)`
- `liquidationCall(address collateralAsset, bool collateralAssetType, address debtAsset, bool debtAssetType, address user,
- `flashLoan(FlashLoanParams memory flashLoanParams, uint256[] calldata amounts, uint256[] calldata modes, bytes cal

AToken

- `transfer(address recipient, uint256 amount)`
- `transferFrom(address sender, address recipient, uint256 amount)`
- `approve(address spender, uint256 amount)`
- `increaseAllowance(address spender, uint256 addedValue)`
- `decreaseAllowance(address spender, uint256 subtractedValue)`
- `permit(address owner, address spender, uint256 value, uint256 deadline, uint8 v, bytes32 r, bytes32 s)`

ATokenNonRebasing

- `transfer(address recipient, uint256 amountShare)`
- `transferFrom(address sender, address recipient, uint256 amountShare)`
- `approve(address spender, uint256 amountShare)`
- `increaseAllowance(address spender, uint256 addedValue)`
- `decreaseAllowance(address spender, uint256 subtractedValue)`

VariableDebtToken

- `approveDelegation(address delegatee, uint256 amount)`

Minipool

- `deposit(address asset, bool wrap, uint256 amount, address onBehalfOf)`
- `withdraw(address asset, bool unwrap, uint256 amount, address to)`
- `borrow(address asset, bool unwrap, uint256 amount, address onBehalfOf)`
- `repay(address asset,  bool wrap, uint256 amount, address onBehalfOf)`
- `setUserUseReserveAsCollateral(address asset, bool useAsCollateral)`
- `liquidationCall(address collateralAsset, address debtAsset, address user, uint256 debtToCover, bool receiveAToken)`
- `flashLoan(FlashLoanParams memory flashLoanParams, uint256[] calldata amounts, uint256[] calldata modes, bytes cal

AToken6909

- `transfer(address to, uint256 id, uint256 amount)`
- `transferFrom(address from, address to, uint256 id, uint256 amount)`
- `approve(address spender, uint256 id, uint256 amount)`
- `setOperator(address operator, bool approved)`
- `approveDelegation(address delegatee, uint256 id, uint256 amount)`

[H-01] `AToken6909.totalSupply()` doesn’t return the correct answer if debtToken.

https://github.com/Cod3x-Labs/Cod3x-Lend/blob/bb2a3c08df5ac7a95ceb20cdccd66c5f007e38d9/contracts/protocol/tokenization/ERC6909/ATokenERC6909.sol#L625

`AToken6909.totalSupply()` doesn't deal with debt tokens.

Call sequence

```
balanceIntegrityMP(): failed!💥
  Call sequence:
    PropertiesMain.randDepositMP((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, false),0,0,0,0,54)
    PropertiesMain.userDebtIntegrityLP() Time delay: 1816 seconds Block delay: 4460
    *wait* Time delay: 23219 seconds Block delay: 2679
    PropertiesMain.randATokenNonRebasingBalanceOfLP((114, 172, 51, 9, 160, 6, 182, 24, 222, 190, 16, 10000000000000
    PropertiesMain.balanceIntegrityMP((17, 146, 190, 120, 135, 223, 0, 48, 56, 52, 6, 4, true)) Time delay: 15120 seconds B
    PropertiesMain.randForceFeedAssetLP((39, 9, 57, 86, 17, 46, 159, 200, 40, 150, 56, 90315082316502981852326478
    PropertiesMain.randFlashloanLP((95, 62, 0, 125, 251, 41, 12, 0, 142, 32, 177, 2304117920, false),48,24,33,1690020659
    PropertiesMain.userDebtIntegrityLP() Time delay: 46623 seconds Block delay: 30
    PropertiesMain.indexIntegrityLP() Time delay: 21088 seconds Block delay: 2135
    PropertiesMain.balanceIntegrityMP((241, 45, 35, 32, 29, 147, 168, 55, 3, 185, 224, 59189340112142212, true))
    PropertiesMain.randATokenNonRebasingBalanceOfLP((121, 41, 101, 31, 153, 21, 19, 16, 6, 56, 32, 224663419882781640
    PropertiesMain.userDebtIntegrityMP() Time delay: 73515 seconds Block delay: 1604
    PropertiesMain.randFlashloanLP((127, 45, 57, 17, 79, 15, 77, 1, 139, 189, 3, 394305762056539185, false),127,50,84,29)
    PropertiesMain.randDepositMP((21, 254, 163, 12, 8, 3, 34, 70, 161, 88, 50, 88, false),14,0,10,48,19421382673352919384
    PropertiesMain.randATokenNonRebasingBalanceOfLP((96, 17, 158, 86, 61, 255, 189, 250, 253, 71, 1, 32228766016004
    PropertiesMain.randForceFeedAssetLP((251, 5, 188, 211, 164, 101, 223, 226, 40, 10, 112, 223, true),192,19,252,31)
    PropertiesMain.indexIntegrityLP()
    *wait* Time delay: 119228 seconds Block delay: 10660
    PropertiesMain.randApproveDelegationMP((97, 2, 125, 34, 2, 102, 127, 44, 171, 95, 236, 134711128324446, true),6,28,6
    *wait* Time delay: 11098 seconds Block delay: 1905
    PropertiesMain.randApproveDelegation((20, 90, 229, 231, 180, 190, 66, 31, 58, 91, 22, 36, true),0,100,39,6201647722
    PropertiesMain.randATokenNonRebasingApproveLP((0, 103, 165, 95, 117, 46, 251, 10, 2, 12, 2, 226956450275534964
    PropertiesMain.randATokenNonRebasingApproveLP((0, 103, 165, 95, 117, 46, 251, 10, 2, 12, 2, 226956450275534964
    PropertiesMain.randApproveMP((91, 63, 2, 128, 72, 46, 209, 65, 139, 20, 74, 256979034224670827132272760797916
    *wait* Time delay: 12080 seconds Block delay: 4446
    PropertiesMain.indexIntegrityLP() Time delay: 13023 seconds Block delay: 5214
    PropertiesMain.randATokenNonRebasingBalanceOfLP((253, 208, 38, 186, 56, 7, 15, 21, 8, 60, 34, 71776119061217282,
    PropertiesMain.randApproveMP((91, 63, 2, 128, 72, 46, 209, 65, 139, 20, 74, 256979034224670827132272760797916
    *wait* Time delay: 55616 seconds Block delay: 6996
    PropertiesMain.randApproveLP((163, 41, 129, 3, 181, 156, 127, 137, 208, 230, 145, 259870524367019817219118179604
    PropertiesMain.randApproveDelegation((51, 84, 166, 35, 59, 99, 224, 36, 0, 76, 63, 492092984003376901729520126
    PropertiesMain.globalSolvencyCheckLP() Time delay: 12756 seconds Block delay: 280
    PropertiesMain.randDepositLP((57, 171, 20, 10, 207, 61, 19, 17, 4, 116, 8, 272310406020363712675636653411240052
    *wait* Time delay: 45606 seconds Block delay: 2912
    PropertiesMain.randApproveMP((91, 63, 2, 128, 72, 46, 209, 65, 139, 20, 74, 256979034224670827132272760797916
    PropertiesMain.randDepositLP((7, 67, 5, 11, 205, 58, 37, 2, 32, 39, 38, 70165120714913858472350060505606445946
    PropertiesMain.userDebtIntegrityMP() Time delay: 85 seconds Block delay: 481
    PropertiesMain.invariantRehypothecationLP()
    PropertiesMain.randFlashloanLP((48, 60, 48, 80, 9, 95, 11, 170, 9, 84, 169, 41502176495236691007300792781773889
    PropertiesMain.balanceIntegrityLP((13, 242, 2, 0, 91, 186, 54, 35, 138, 163, 180, 20663932105955436224767242542
    PropertiesMain.randRehypothecationRebalanceLP((0, 17, 0, 118, 1, 0, 1, 12, 29, 91, 3, 20864817560845552097478642
    PropertiesMain.randFlashloanLP((17, 191, 86, 20, 7, 34, 252, 89, 61, 249, 36, 164211499452241065304650123
    *wait* Time delay: 47389 seconds Block delay: 5321
```

```
PropertiesMain.randForceFeedAssetLP((78, 207, 202, 243, 13, 25, 164, 65, 207, 194, 154, 77, true),51,3252853869160
PropertiesMain.randATokenNonRebasingApproveLP((0, 103, 72, 45, 117, 46, 48, 10, 0, 11, 1, 22695645027553496403
PropertiesMain.userDebtIntegrityMP()
*wait* Time delay: 48 seconds Block delay: 2971
*wait* Time delay: 74283 seconds Block delay: 4045
PropertiesMain.randRehypothecationRebalanceLP((159, 235, 3, 80, 54, 44, 48, 35, 97, 207, 57, 192481140412600448
PropertiesMain.randApproveLP((79, 23, 63, 21, 59, 39, 216, 253, 87, 159, 171, 30827016758345989097485660825630
PropertiesMain.randApproveLP((231, 63, 155, 165, 132, 164, 249, 228, 11, 253, 5, 47, true),254,16,161,667445711395516
PropertiesMain.randRehypothecationRebalanceLP((108, 19, 8, 6, 0, 9, 92, 32, 11, 79, 41, 10499, false),0)
*wait* Time delay: 162891 seconds Block delay: 27233
PropertiesMain.randApproveDelegation((81, 63, 161, 16, 22, 170, 207, 52, 254, 35, 161, 1774647075, true),33,53,32,765
PropertiesMain.randDepositLP((60, 253, 65, 130, 108, 21, 105, 17, 220, 223, 104, 8501, false),129,4,53,138) Time delay:
PropertiesMain.randApproveMP((121, 97, 38, 70, 17, 103, 249, 65, 79, 2, 47, 125371356139757617497740711158745669
PropertiesMain.randRehypothecationRebalanceLP((161, 85, 85, 247, 12, 155, 252, 253, 129, 158, 207, 11906361879543
PropertiesMain.balanceIntegrityMP((11, 194, 97, 0, 53, 56, 20, 105, 194, 27, 20, 89920122055789005477494441605795
PropertiesMain.randApproveDelegation((21, 4, 52, 223, 1, 1, 9, 163, 36, 147, 31, 190570488831882540329701954132 6
*wait* Time delay: 203494 seconds Block delay: 6047
PropertiesMain.userDebtIntegrityMP()
PropertiesMain.userDebtIntegrityLP()
*wait* Time delay: 30845 seconds Block delay: 4305
PropertiesMain.userDebtIntegrityMP() Time delay: 78 seconds Block delay: 2948
PropertiesMain.balanceIntegrityMP((77, 8, 32, 33, 4, 24, 10, 6, 145, 29, 0, 8, false))
*wait* Time delay: 86037 seconds Block delay: 7373
PropertiesMain.randATokenNonRebasingApproveLP((0, 103, 165, 95, 117, 46, 251, 10, 2, 12, 2, 226956450275534964(
PropertiesMain.randDepositMP((46, 224, 184, 84, 5, 127, 178, 45, 253, 254, 5, 223, false),155,140,39,68,33540519)
PropertiesMain.globalSolvencyCheckLP() Time delay: 49950 seconds Block delay: 2948
PropertiesMain.randATokenNonRebasingApproveLP((77, 97, 13, 49, 253, 35, 21, 99, 68, 83, 95, 264051319282080588
PropertiesMain.randForceFeedAssetLP((121, 196, 95, 129, 47, 47, 84, 133, 53, 68, 93, 28012052341044330094712598
PropertiesMain.globalSolvencyCheckLP()
PropertiesMain.randDepositMP((50, 236, 34, 13, 177, 63, 33, 231, 142, 160, 247, 100000000, false),161,0,213,69,45999
PropertiesMain.randApproveLP((66, 33, 10, 92, 113, 65, 76, 133, 144, 93, 48, 31784581303690635858714900557967(
*wait* Time delay: 61634 seconds Block delay: 7197
PropertiesMain.indexIntegrityLP()
PropertiesMain.randATokenNonRebasingBalanceOfLP((223, 73, 219, 50, 1, 17, 1, 3, 6, 24, 1, 146, false),2,10)
PropertiesMain.randATokenNonRebasingApproveLP((8, 76, 86, 118, 94, 253, 46, 14, 92, 15, 77, 1527498425753102410
PropertiesMain.globalSolvencyCheckMP()
PropertiesMain.randApproveLP((8, 254, 13, 166, 164, 47, 21, 136, 7, 89, 252, 258158516, true),237,253,14,2489636238
PropertiesMain.randSetUseReserveAsCollateralMP((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 21846270404136121375367471326 3
PropertiesMain.globalSolvencyCheckMP() Time delay: 1426 seconds Block delay: 4387
PropertiesMain.userDebtIntegrityLP() Time delay: 23959 seconds Block delay: 5637
*wait* Time delay: 80900 seconds Block delay: 12739
PropertiesMain.userDebtIntegrityMP() Time delay: 32745 seconds Block delay: 1846
PropertiesMain.randATokenNonRebasingBalanceOfLP((241, 233, 254, 7, 41, 18, 4, 92, 164, 46, 4, 3402823669209384
PropertiesMain.randApproveMP((91, 63, 2, 128, 72, 46, 209, 65, 139, 20, 74, 25697903422467082713227276079791 6
*wait* Time delay: 31346 seconds Block delay: 7468
PropertiesMain.randATokenNonRebasingApproveLP((252, 162, 63, 36, 161, 40, 76, 198, 110, 212, 93, 0, false),170,72,5,
*wait* Time delay: 23845 seconds Block delay: 6913
PropertiesMain.randApproveDelegation((209, 254, 76, 87, 252, 80, 153, 21, 192, 18, 170, 18, false),10,236,7,502) Time d
*wait* Time delay: 137297 seconds Block delay: 3539
*wait* Time delay: 64670 seconds Block delay: 781
PropertiesMain.randApproveDelegation((20, 92, 31, 53, 56, 37, 187, 16, 165, 10, 148, 5, false),152,208,182,2978510941 6
PropertiesMain.userDebtIntegrityLP() Time delay: 68550 seconds Block delay: 5000
PropertiesMain.randApproveDelegation((13, 161, 3, 34, 44, 85, 1, 64, 229, 21, 9, 8000, false),231,20,69,1534175515644
PropertiesMain.randBorrowMP((246, 132, 94, 225, 62, 252, 136, 230, 6, 252, 32, 68143482658907150828985204057
```

```
    PropertiesMain.randForceFeedAssetLP((0, 45, 26, 60, 209, 14, 0, 227, 51, 3, 9, 753639622553312597832086281656
    PropertiesMain.invariantRehypothecationLP()
    PropertiesMain.randATokenNonRebasingBalanceOfLP((83, 9, 81, 5, 9, 25, 62, 15, 9, 253, 254, 2645899438, false),197
    *wait* Time delay: 16424 seconds Block delay: 5721
    PropertiesMain.invariantRehypothecationLP() Time delay: 80854 seconds Block delay: 626
    PropertiesMain.randApproveMP((21, 7, 8, 31, 0, 146, 83, 45, 28, 16, 107, 124515559289431977190045388494240733
    PropertiesMain.randATokenNonRebasingApproveLP((22, 4, 86, 156, 254, 136, 33, 254, 246, 251, 203, 783, false),186,
    PropertiesMain.randApproveMP((91, 63, 2, 128, 48, 29, 141, 65, 39, 20, 74, 256216505162522495745240811746225
    PropertiesMain.globalSolvencyCheckMP()
    PropertiesMain.randDepositMP((186, 238, 24, 0, 110, 32, 93, 57, 98, 153, 1, 114725417418437506094518906338289
    PropertiesMain.globalSolvencyCheckMP()
    PropertiesMain.randIncreaseAllowanceLP((56, 8, 240, 79, 206, 234, 65, 78, 59, 52, 251, 33540519, false),7,2,48,486
    PropertiesMain.invariantRehypothecationLP()
    PropertiesMain.randDepositLP((8, 19, 101, 6, 75, 98, 163, 250, 91, 56, 180, 161952454400808545378709638062577
    *wait* Time delay: 35918 seconds Block delay: 7639
    PropertiesMain.indexIntegrityLP()
    *wait* Time delay: 188294 seconds Block delay: 3585
    PropertiesMain.randRehypothecationRebalanceLP((205, 223, 37, 129, 56, 27, 249, 127, 150, 32, 148, 17953301117148
    PropertiesMain.randApproveMP((137, 63, 2, 44, 72, 42, 83, 65, 139, 20, 74, 23435789016932680730615880247832
    PropertiesMain.randApproveMP((4, 163, 22, 183, 160, 169, 73, 65, 7, 214, 38, 768, false),22,48,0,111,843049181685847
    PropertiesMain.randATokenNonRebasingApproveLP((246, 119, 79, 6, 254, 173, 33, 1, 64, 184, 63, 65537, true),0,85,25
    PropertiesMain.randBorrowMP((76, 234, 88, 45, 251, 125, 101, 216, 184, 142, 113, 11935599927562338104773530395
    PropertiesMain.randATokenNonRebasingApproveLP((0, 144, 165, 95, 117, 46, 222, 6, 0, 2, 0, 8294676312090815016
    *wait* Time delay: 53014 seconds Block delay: 3457
    PropertiesMain.globalSolvencyCheckLP()
    PropertiesMain.globalSolvencyCheckMP()
    *wait* Time delay: 5820 seconds Block delay: 8440
    PropertiesMain.randApproveDelegation((1, 31, 213, 154, 95, 159, 63, 252, 252, 74, 95, 91355125619628245492144631
    PropertiesMain.randApproveDelegationMP((200, 224, 61, 145, 96, 229, 165, 49, 122, 7, 91, 99999, false),53,222,20,4)
    PropertiesMain.randApproveMP((91, 5, 159, 151, 51, 197, 67, 253, 197, 65, 18, 29950613002632017509409596699997
    PropertiesMain.balanceIntegrityMP((7, 254, 62, 53, 127, 216, 236, 191, 52, 30, 76, 20, true)) Time delay: 21372 second

    emit AssertFail(«617»)
```

Recommendation

from:

```
function totalSupply(uint256 id) public view override returns (uint256) {
  uint256 currentSupplyScaled = super.totalSupply(id);

  if (currentSupplyScaled == 0) {
    return 0;
  }

  return currentSupplyScaled.rayMul(
    POOL.getReserveNormalizedIncome(_underlyingAssetAddresses[id])
  );
}
```

To:

```
function totalSupply(uint256 id) public view override returns (uint256) {
  uint256 currentSupplyScaled = super.totalSupply(id);
```

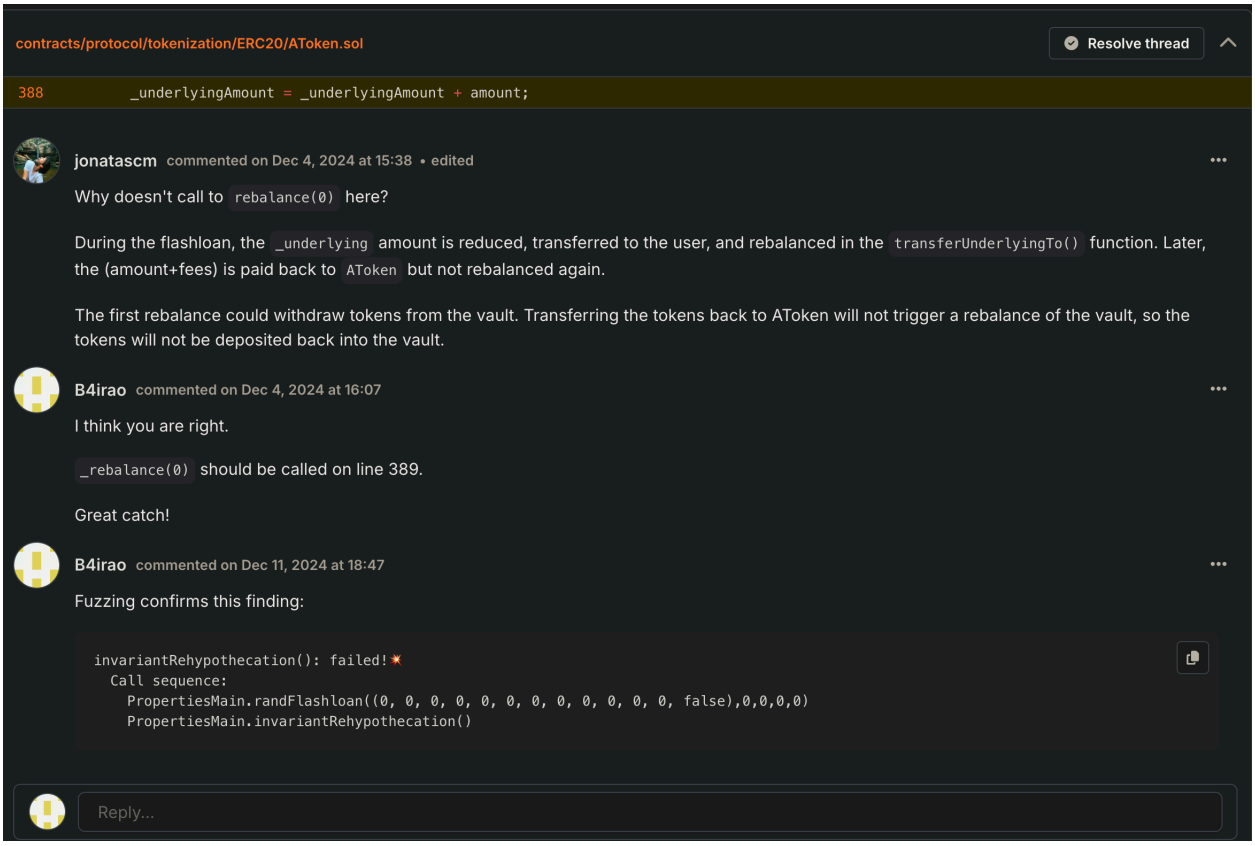
```
    if (currentSupplyScaled == 0) {
      return 0;
    }

    if (isDebtToken(id)) {
      return currentSupplyScaled.rayMul(
        POOL.getReserveNormalizedVariableDebt(_underlyingAssetAddresses[id])
      );
    } else {
      return currentSupplyScaled.rayMul(
        POOL.getReserveNormalizedIncome(_underlyingAssetAddresses[id])
      );
    }
  }
```

Fix :: https://github.com/Cod3x-Labs/Cod3x-Lend/commit/f8cd1275b4764da728e040ed9dcfb0bcf314b332

[H-02] Rebalance rehypothecation DOS

https://github.com/Cod3x-Labs/Cod3x-Lend/blob/bb2a3c08df5ac7a95ceb20cdccd66c5f007e38d9/contracts/protocol/tokenization/ERC20/AToken.sol#L383-L389



Call sequence

```
invariantRehypothecation(): failed!💥
  Call sequence:
    PropertiesMain.randFlashloan((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, false),0,0,0,0)
    PropertiesMain.invariantRehypothecation()
```

Recommendation

`_rebalance(0)` must be added to `handleRepayment()` and the rehypothecation logic must be non-blocking to avoid any DOS associated with the rehypothecation vault.

Fix ::: https://github.com/Cod3x-Labs/Cod3x-Lend/commit/c5b28c181ffee86d85480a49b1d88251ab7eaf92/contracts/protocol/tokenization/ERC20/AToken.sol#diff-87e27a6bc579861af841bca6e0a97d87a90079147b7ca9596be0be0e041340dc

[M-01] Rounding issue in `ValidationLogic.validateWithdraw()`

https://github.com/Cod3x-Labs/Cod3x-Lend/blob/bb2a3c08df5ac7a95ceb20cdccd66c5f007e38d9/contracts/protocol/core/lendingpool/logic/GenericLogic.sol#L1L140

https://github.com/Cod3x-Labs/Cod3x-Lend/blob/bb2a3c08df5ac7a95ceb20cdccd66c5f007e38d9/contracts/protocol/core/minipool/logic/MiniPoolGenericLogic.sL124

Properties `230` and `505` failed because the way the health factor is calculated in `ValidationLogic.validateWithdraw()` is different from the actual health factor at the end of the withdrawal transaction.

The health factor is greater than `1e18` here, but ends up being less once the transaction is complete.

Call sequence

224 - ❌ `withdraw()` must not result in a health factor of less than 1.

```
randWithdrawMP((uint8,uint8,uint8,uint8,uint8,uint8,uint8,uint8,uint8,uint8,uint8,uint128,bool),uint8,uint8,uint8,uint8,uint12
  Call sequence:
    PropertiesMain.randDepositMP((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 55231321179, false),0,0,2,0,1)
    PropertiesMain.randBorrowMP((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, false),0,0,2,1,0)
    PropertiesMain.randWithdrawMP((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 23295806495239189076 9, false),0,2,2,0,1)

emit AssertFail(«224»)
```

505 - ❌ `withdraw()` must not result in a health factor of less than 1.

```
  randWithdrawMP((uint8,uint8,uint8,uint8,uint8,uint8,uint8,uint8,uint8,uint8,uint8,uint128,bool),uint8,uint8,uint8,uint8,uint1
  Call sequence:
    PropertiesMain.randDepositMP((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 55231321179, false),0,0,2,0,1)
    PropertiesMain.randBorrowMP((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, false),0,0,2,1,0)
    PropertiesMain.randWithdrawMP((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 23295806495239189076 9, false),0,2,2,0,1)

    emit AssertFail(«505»)
```

Recommendation

From:

```
function getAmountToDecreaseInEth(
    address oracle,
    address asset,
    uint256 amount,
    uint256 decimals
) internal view returns (uint256) {
    return IOracle(oracle).getAssetPrice(asset) * amount / (10 ** decimals);
}
```

To:

```
function getAmountToDecreaseInEth(
    address oracle,
    address asset,
    uint256 amount,
    uint256 decimals
) internal view returns (uint256) {
    return WadRayMath.divUp(IOracle(oracle).getAssetPrice(asset) * amount,10 ** decimals);
}
```

In both `GenericLogic` and `MinipoolGenericLogic`.

Fix ::: https://github.com/Cod3x-Labs/Cod3x-Lend/commit/c72d768bd55fa969fad1d25af3930e1b7d87ae9e

[M-02] Rounding issue in `ValidationLogic.validateBorrow()`

https://github.com/Cod3x-Labs/Cod3x-Lend/blob/bb2a3c08df5ac7a95ceb20cdccd66c5f007e38d9/contracts/protocol/core/lendingpool/logic/ValidationLogic.sol#

https://github.com/Cod3x-Labs/Cod3x-Lend/blob/bb2a3c08df5ac7a95ceb20cdccd66c5f007e38d9/contracts/protocol/core/minipool/logic/MiniPoolValidationLogic

Properties `207` and `508` failed because the way the health factor is calculated in `ValidationLogic.validateWithdraw()` is different from the actual health factor at the end of the withdrawal transaction.

The health factor is greater than `1e18` here, but ends up being less once the transaction is complete.

Call sequence

207 - ❌ `borrow()` must not result in a health factor of less than 1.

```
randBorrowLP((uint8,uint8,uint8,uint8,uint8,uint8,uint8,uint8,uint8,uint8,uint8,uint128,bool),uint8,uint8,uint8,uint128): failed
  Call sequence:
    PropertiesMain.randApproveDelegation((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, false),0,0,0,0)
    PropertiesMain.randApproveDelegation((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, false),0,0,0,0)
    PropertiesMain.randATokenNonRebasingBalanceOfLP((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, false),0,0)
    PropertiesMain.randDepositLP((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 15989, false),0,0,1,50)
    PropertiesMain.randFlashloanLP((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, false),0,0,0,0)
    PropertiesMain.randApproveMP((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, false),0,0,0,0,0)
    PropertiesMain.randForceFeedAssetLP((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, false),0,0,0,0)
    PropertiesMain.randATokenNonRebasingBalanceOfLP((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, false),0,0)
    PropertiesMain.randATokenNonRebasingBalanceOfLP((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, false),0,0)
    PropertiesMain.randApproveDelegation((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, false),0,0,0,0)
    PropertiesMain.randFlashloanLP((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, false),0,0,0,0)
```

```
    PropertiesMain.randBorrowLP((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, false),0,0,0,3)

emit AssertFail(«207»)
```

Recommendation

From:

```
require(
    vars.amountOfCollateralNeededETH <= vars.userCollateralBalanceETH,
    Errors.VL_COLLATERAL_CANNOT_COVER_NEW_BORROW
);
```

To:

```
require(
    vars.amountOfCollateralNeededETH < vars.userCollateralBalanceETH,
    Errors.VL_COLLATERAL_CANNOT_COVER_NEW_BORROW
);
```

In both `ValidationLogic` and `MinipoolValidationLogic`.

Fix ::: https://github.com/Cod3x-Labs/Cod3x-Lend/commit/72a1dfc703d81acb093744c04003b3b21174b341

[M-03] `userConfig` inconsistency

https://github.com/Cod3x-Labs/Cod3x-Lend/blob/bb2a3c08df5ac7a95ceb20cdccd66c5f007e38d9/contracts/protocol/core/lendingpool/logic/LiquidationLogic.sol

https://github.com/Cod3x-Labs/Cod3x-Lend/blob/bb2a3c08df5ac7a95ceb20cdccd66c5f007e38d9/contracts/protocol/core/minipool/logic/MiniPoolLiquidationLog

`userConfig.isUsingAsCollateral()` returns `false` even if the user has collateral.

Condition:

- Call `liquidationCall()` with the `Liquidator` and the `User` liquidated are the same
- `receiveAToken == true`
- `vars.maxCollateralToLiquidate == vars.userCollateralBalance`

These condition will write false in the userConfig for the given collateral address. But because the `Liquidator` and the `User` are the same, the user will end up with aTokens but `userConfig.isUsingAsCollateral()` will return `false`.

Call sequence

521 - ❌ `UserConfigurationMap` integrity: If a user has a given aToken then `isUsingAsCollateralOrBorrowing` and `isUsingAsCollateral` should return true.

```
userConfigurationMapIntegrityLiquidityMP(): failed!💥
  Call sequence:
    PropertiesMain.randForceFeedAssetLP((5, 152, 128, 48, 100, 9, 2, 40, 34, 8, 34, 1501, false),121,54226010652114989114
    PropertiesMain.randRehypothecationRebalanceLP((0, 97, 131, 3, 114, 9, 18, 69, 14, 12, 0, 64414634520032050944263855
    PropertiesMain.balanceIntegrityMP((3, 147, 16, 52, 40, 1, 41, 224, 2, 0, 6, 94498613017923544327947094807073594 6,
    PropertiesMain.randRehypothecationRebalanceLP((7, 16, 3, 19, 3, 93, 0, 0, 21, 0, 1, 82787515665385814594281779132 5
    PropertiesMain.balanceIntegrityMP((7, 21, 4, 4, 3, 44, 2, 98, 26, 24, 1, 29936668805365269966752080644305993363
    PropertiesMain.randForceFeedAssetLP((8, 41, 53, 10, 4, 146, 2, 52, 4, 63, 4, 23041274427037666899071109915929515
```

```
  PropertiesMain.randATokenNonRebasingBalanceOfLP((50, 33, 62, 30, 11, 69, 0, 85, 12, 1, 0, 709682467078502831047 5
  PropertiesMain.randATokenNonRebasingBalanceOfLP((13, 214, 12, 0, 55, 97, 6, 11, 3, 24, 2, 22311466253387933517001 9
  PropertiesMain.randRehypothecationRebalanceLP((0, 5, 2, 0, 0, 19, 1, 17, 15, 0, 1, 153990134152141945583668968611 80
  PropertiesMain.randApproveDelegationMP((7, 7, 136, 7, 2, 27, 37, 16, 188, 164, 155, 125457204010425726085156370230
  PropertiesMain.randApproveMP((141, 53, 6, 15, 3, 46, 2, 0, 26, 3, 32, 93, false),0,9,0,24,7)
  PropertiesMain.randForceFeedAssetLP((1, 4, 20, 1, 0, 0, 1, 4, 1, 0, 0, 77999725134659503690874698400990357, false),
  PropertiesMain.randIncreaseAllowanceLP((27, 225, 28, 58, 0, 9, 144, 45, 6, 64, 99, 2000, false),13,1,0,26)
  PropertiesMain.randApproveDelegation((55, 42, 157, 84, 156, 23, 20, 28, 232, 57, 92, 53138198135018813486024556, fa
  PropertiesMain.randFlashloanLP((19, 173, 19, 24, 13, 2, 4, 30, 42, 222, 89, 56298442179623642784898957328763779 7
  PropertiesMain.randFlashloanLP((10, 224, 85, 153, 63, 89, 17, 211, 53, 8, 5, 22794679257771908030857608548902800 1
  PropertiesMain.randATokenNonRebasingBalanceOfLP((116, 6, 251, 2, 201, 75, 0, 0, 25, 0, 2, 893778136417665197399, fa
  PropertiesMain.randDepositLP((37, 0, 85, 159, 18, 21, 18, 46, 0, 25, 14, 19517060225048056303761628647016002362, f
  PropertiesMain.randATokenNonRebasingApproveLP((86, 4, 52, 0, 4, 3, 0, 51, 57, 77, 11, 42140458220383639004056170
  PropertiesMain.randIncreaseAllowanceLP((210, 37, 48, 104, 29, 114, 1, 45, 215, 54, 86, 76584090283686923044706425
  PropertiesMain.randIncreaseAllowanceLP((17, 14, 70, 115, 34, 241, 57, 58, 145, 205, 239, 384000, false),0,9,86,3)
  PropertiesMain.randATokenNonRebasingBalanceOfLP((2, 2, 131, 2, 41, 57, 0, 6, 3, 4, 14, 2054715664365430604936, fal
  PropertiesMain.balanceIntegrityLP((5, 5, 23, 4, 26, 0, 0, 49, 0, 117, 131, 92800218047652643161337046704555957618, 1
  PropertiesMain.randATokenNonRebasingApproveLP((1, 29, 0, 16, 129, 4, 237, 62, 117, 25, 59, 63759371, false),2,91,120,7 0
  PropertiesMain.randDepositMP((161, 48, 156, 7, 3, 44, 38, 18, 78, 178, 31, 65536, true),0,8,5,85,27941694593782908584
  PropertiesMain.randApproveMP((103, 75, 80, 15, 6, 99, 34, 92, 0, 168, 2, 6466, false),5,1,0,12,2168789229610216787149 4
  PropertiesMain.randIncreaseAllowanceLP((3, 0, 14, 4, 1, 10, 204, 0, 87, 2, 22, 1057948505, false),0,0,1,193214815738136
  PropertiesMain.randApproveDelegation((81, 15, 53, 249, 97, 8, 1, 0, 174, 48, 8, 254770957443543333949829060447844
  PropertiesMain.randApproveDelegation((0, 8, 254, 1, 3, 18, 6, 7, 54, 20, 54, 22324779232128438420906072263032275
  PropertiesMain.randRehypothecationRebalanceLP((5, 21, 82, 22, 9, 111, 53, 1, 0, 0, 1, 38348043, false),0)
  PropertiesMain.balanceIntegrityMP((0, 1, 5, 3, 5, 59, 0, 0, 0, 2, 17, 92008150300111475425814864436454221 90, false))
  PropertiesMain.randBorrowMP((18, 174, 57, 15, 31, 27, 52, 148, 179, 112, 0, 3858086692, false),4,1,49,226,218343749412
  PropertiesMain.randATokenNonRebasingBalanceOfLP((0, 1, 153, 0, 11, 4, 2, 26, 3, 0, 0, 2833135388202602584488, fals
  PropertiesMain.randApproveDelegationMP((4, 252, 163, 41, 248, 1, 243, 5, 43, 18, 27, 123825494993740765743949008
  PropertiesMain.randDepositMP((135, 225, 46, 84, 33, 65, 217, 70, 75, 121, 88, 340282366920938463463374607431768
  PropertiesMain.randDepositLP((0, 179, 164, 0, 0, 0, 67, 10, 20, 61, 6, 11130080049166954077087256390325995 26, false
  PropertiesMain.randApproveLP((4, 0, 190, 5, 21, 35, 91, 0, 3, 10, 0, 369009189304850255002792339468318790 92, fals
  PropertiesMain.randDepositMP((89, 68, 2, 0, 27, 159, 94, 21, 177, 19, 7, 14037164304540552964801149035513826379 1,
  PropertiesMain.userConfigurationMapIntegrityLiquidityMP()

 emit AssertFail(«521»)
```

Recommendation

From:

```
if (vars.maxCollateralToLiquidate == vars.userCollateralBalance) {
    userConfig.setUsingAsCollateral(collateralReserve.id, false);
    emit ReserveUsedAsCollateralDisabled(params.collateralAsset, params.user);
}
```

To:

```
if (vars.collateralAtoken.balanceOf(params.user) == 0) {
    userConfig.setUsingAsCollateral(collateralReserve.id, false);
    emit ReserveUsedAsCollateralDisabled(params.collateralAsset, params.user);
}
```

In both `LiquidationLogic` and `MiniPoolLiquidationLogic`.

Fix ::: https://github.com/Cod3x-Labs/Cod3x-Lend/commit/ea09026fe0f770e1cbc90f6ef0d12d069c55e7df