# Cod3x LEND 1 - Beirao audit 31/08/2024

## 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.

This audit was conducted in conjunction with Rave110X.

## 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.

### Privileged Roles & Actors

Same as AAVE V2 roles.

### Previous audits

*Trail of Bits:*

Cod3x Foundation - Cod3x Lend - Comprehensive Security Assessment.pdf

## 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*** **-** b007943c

***fixes review commit hash*** **-** 866da3d5

## Deployment chains

- All EVMs.

## Scope

The following smart contracts were in scope of the audit: (total : **9921** SLoC)

- `contracts/**`

# Findings Summary

Summary :

- **3** Critical(s)
- **3** High(s)
- **1** Medium(s)
- **0** Low(s)
- **10** Informational(s)
- **1** Undetermined

| ID | Title | Status |
|---|---|---|
| [C-01] | Rehypothecation breaks aToken internal accounting | Fix |
| [C-02] | AAVE doesn't support rebasing tokens natively, but main `lendingPool` aToken are rebasing | Fix |
| [C-03] | Main `lendingPool` flashloan feature breaks internal accounting due to rehypothecation. | Fix |
| [H-01] | `lendingPool` variables can't be set | Fix |
| [H-02] | Minipool `_miniPoolToTreasury` can't be set | Fix |
| [H-03] | `updateState()` not called when minipool borrow | Fix |
| [M-01] | Minipool borrows and flow limiter concerns | Fix |
| [U-01] | Augmented interest rate logic never reached | Fix |
| [I-01] | Bad UX on tranched borrows | Ack |
| [I-02] | Remove any `reserveType` variable from mini pool code. | Ack |
| [I-03] | No permit on `AERC6909` | Ack |
| [I-04] | `ERC6909` does not implement the rewarder | Fix |
| [I-05] | Some of the natspecs are not up to date, especially for the minipool logic | Fix |
| [I-06] | Remove all `console.log` before production | Fix |

| ID | Title | Status |
|---|---|---|
| [I-07] | Remove all useless commented code | Fix |
| [I-08] | Variable `availableMLPLiquidity` always 0 and it is unused | Fix |
| [I-09] | Some errors are not properly defined and need a description. (ex: `revert()`) | Fix |
| [I-10] | Some zero input checks may be missing (ex: `LendingPoolAddressesProvider.setAddress()`) | Fix |

# Detailed Findings

# [C-01] Rehypothecation breaks aToken internal accounting

https://github.com/Cod3x-Labs/Cod3x-Lend/blob/b007943c2a08d5431c0bde56ec08687ff45903a9/contracts/protocol/tokenization/AToken.sol#L376-L380

## Severity

**Impact:** High, break liquidity accounting.

**Likelihood:** High, happens everytime at debt repayment.

## Description

Because of rehypothecation we now use a storage variable (`underlyingAmount`) to replace `asset.balanceOf(aToken)`.
Problem : When we transfer asset (borrow) we correctly update this var but not when we repay the loan.

## Recommendations

```
    function handleRepayment(address user, address onBehalfOf, uint256 amount)
        external
        override
        onlyLendingPool
    {
+       underlyingAmount = underlyingAmount.add(amount);
    }
```

# [C-02] AAVE doesn't support rebasing tokens natively, but main lendingpool aToken are rebasing

https://governance.aave.com/t/ampl-problem-on-aave-v2-ethereum/15886

## Severity

**Impact:** High, lost of fund and unexpected behaviours.

**Likelihood:** High, all 'tranched' asset operation on minipools are affected.

## Description

I think aTokens are not compatible because minipool will mint at t0 "amount" == "amount aToken" == "amount aaToken" here. And then at t1 user will burn this "amount" without accounting for the aToken rebasing here.

⇒ I think the result is interests earned between t0 and t1 by aToken will be stuck in the minipool (edited)

(`Flowlimiter.currentFlow()` don't account for interests rate)

## Recommendations

Build a aToken wrapper to make it non rebasing. This way, augmented interest rate logic can be removed.

Issues

## Evidences

- **Evidence 1 - Additiveness check shows some rounding issues**
  - Occurred in test: testErc6909TransferFrom_AToken, testErc6909TransferOnLiquidation_AToken
  - Short description: When there is a different index due to value appreciation during time, the transfers are not properly calculated
  - Link
  - **Severity: Medium**
    - **Likelihood: High - everytime aToken appreciated in value and somebody wants to transfer from aTokens**
    - **Impact: Low - it is probably just the precision issue which must be investigated**

**Recommendation:**

Investigate calculations and rounding and commonize it between tests and implementation

- **Evidence 2 - Users cannot withdraw all funds after borrow and repay (after some time) in mini pools**
  - **Occurred in test:** testMultipleUsersBorrowRepayAndWithdraw
  - **Short description:** Users are not able to withdraw all funds that they deposited after repaying
  - **Severity: High**
    - **Likelihood: High - basic actions after a certain period of time**
    - **Impact: High - users are not able to withdraw the greater or equal amount of funds that they deposited**

**Recommendation:**

Needs further investigation

# [C-03] Main lendingPool flashloan feature breaks internal accounting due to rehypothecation.

https://github.com/Cod3x-Labs/Cod3x-Lend/blob/89504afdc4282c10d326cf562a57b8711d375a8f/contracts/protocol/libraries/logic/FlashLoanLogic.sol

## Severity

**Impact:** High, breaks liquidity accounting.

**Likelihood:** High, happens on every flashloans.

## Description

As you can see FlashLoanLogic uses `IAToken(aTokenAddresses[i]).transferUnderlyingTo()`.

`transferUnderlyingTo` updates `underlyingAmount` (here) but FlashLoanLogic just send back funds without calling handleRepayment.

By not calling handleRepayment(), the flashloan feature decrease `underlyingAmount` but don't increase it when repaying the flashloan. This is messing up the accounting of the Atoken.

## Recommendations

Cod3x LEND must get inspired by the AAVE V3 flashloan implementation that fixes this.

https://github.com/aave/aave-v3-core/blob/b74526a7bc67a3a117a1963fc871b3eb8cea8435/contracts/protocol/libraries/logic/FlashLoanLogic.sol#L254-L258

# [H-01] `lendingPool` variables can't be set

https://github.com/Cod3x-Labs/Cod3x-Lend/blob/b007943c2a08d5431c0bde56ec08687ff45903a9/contracts/protocol/lendingpool/LendingPool.sol#L752-L776

## Severity

**Impact**: Medium, these parameters cannot be changed. A lendingPool update will be required.

**Likelihood:** High, DAO will need to call these settings sooner or later.

## Description

`updateFlashLoanFee()`, `setRewarderForReserve()` and `setTreasury()` are defined in LendingPool, but lendingPoolConfigurator is not updated to support these new setters. So these 3 functions can't be called.

## Recommendations

LendingPoolConfigurator needs to be updated. Some other setters may be affected by this bug. Please make sure that this bug is not in any other part of the code.

# [H-02] minipool `_miniPoolToTreasury` can't be set

https://github.com/Cod3x-Labs/Cod3x-Lend/blob/ef2880fd6e8bf75f43de7a58a077c35942eab0c7/contracts/protocol/configuration/MiniPoolAddressProvider.sol#l

## Severity

**Impact**: Medium, `_miniPoolToTreasury` will always be `address(0)` so reserve factor will not be able to collect fees. A lendingPool update will be required.

**Likelihood:** High, DAO will need to call these settings sooner or later.

## Description

`_miniPoolToTreasury` will always be `address(0)` so reserve factor will not be able to collect fees. A lendingPool update will be required.

## Recommendations

Add a setter for `_miniPoolToTreasury`.

# [H-03] `updateState()` not called when minipool borrow.

https://github.com/Cod3x-Labs/Cod3x-Lend/blob/89504afdc4282c10d326cf562a57b8711d375a8f/contracts/protocol/libraries/logic/BorrowLogic.sol#L423

## Severity

**Impact:** Medium, only a problem if the state has not been updated for a while.

**Likelihood:** High, every time the minipool borrows.

## Description

Minipool doesn’t update the index when borrowing. A non-updated index when minting debtTokens makes the

## Recommendations

Add `updateState()` in executeMiniPoolBorrow.

# [M-01] Minipool borrows and flow limiter concerns.

https://github.com/Cod3x-Labs/Cod3x-Lend/blob/89504afdc4282c10d326cf562a57b8711d375a8f/contracts/protocol/lendingpool/minipool/FlowLimiter.sol

https://github.com/Cod3x-Labs/Cod3x-Lend/blob/89504afdc4282c10d326cf562a57b8711d375a8f/contracts/protocol/lendingpool/minipool/MiniPoolDefaultReserve

## Severity

**Impact:** Undefined.

**Likelihood:** High, every lending pool operations.

## Description

**Bad utilization rate calculation on interests rate default strategy.**

This line is correct only if the asset.isTranche() == false. If the asset is borrowed from the main lending pool, then we need to read the aToken from the main lending pool flowLimiter adjusted.

In this setup, minipool borrows from main lendingpool will happen at 100% interests rate.

**Flow limiter may have bad outcomes**

Otherwise, the risk is not fully isolated and bad debts from minipools can infect the main credit pool.

## Recommendations

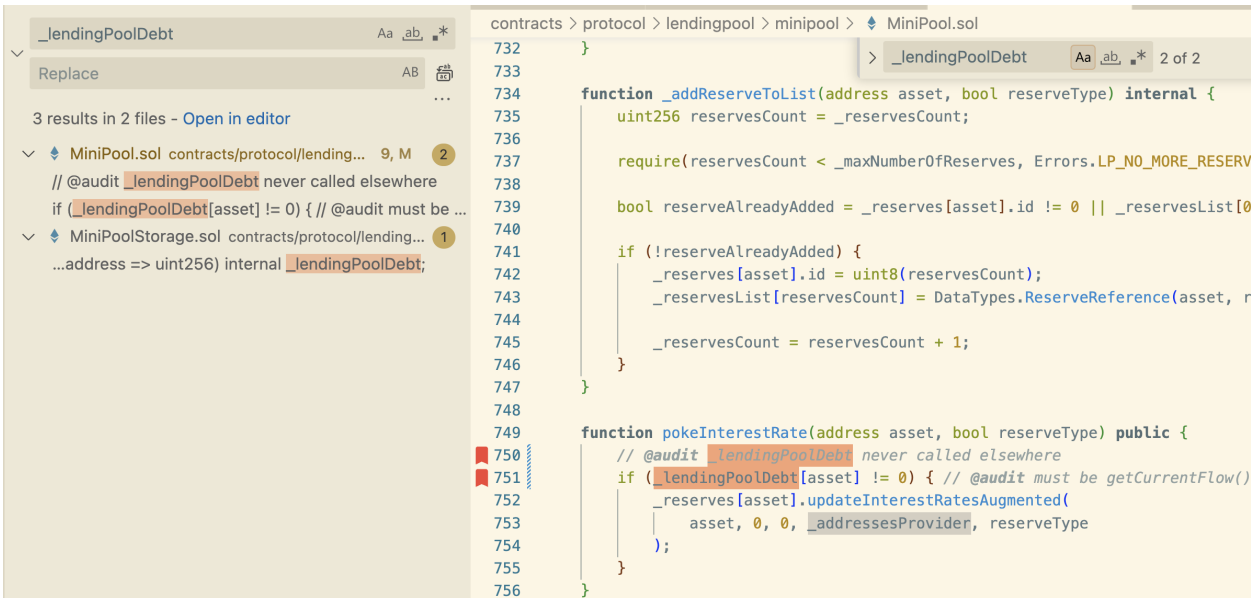
Add checks that limit the minipool borrow to a safe value and define the correct behaviour the strategy should have when reaching 100% utilization.

# [U-01] Augmented interest rate logic never reached

https://github.com/Cod3x-Labs/Cod3x-Lend/blob/89504afdc4282c10d326cf562a57b8711d375a8f/contracts/protocol/lendingpool/minipool/MiniPool.sol#L750-L754

## Description

`_lendingPoolDebt` is never assigned so updateInterestRatesAugmented is never called.




# Quality control

## [I-01] Bad UX on tranched borrows

https://github.com/Cod3x-Labs/Cod3x-Lend/blob/89504afdc4282c10d326cf562a57b8711d375a8f/contracts/protocol/tokenization/ERC6909/ATokenERC6909.sol#

## Description

To get the underlying token when borrowing from minipool (if isTranche == true); the user will need to call `minipool.borrow()` and `mainpool.withdraw()` which is not ideal for UX.

And then to repay: `asset.approve()` ⇒ `mainpool.deposit()` ⇒ `minipool.repay()`

## Recommendations

Replacing this transfer() by `withdraw()` :

```
mainLendingPool.withdraw(IAToken(underlyingAssetAddresses[id]).UNDERLYING_ASSET_ADDRESS(), tru
e, amount, to);
```

## [I-02] Remove any `reserveType` variable from mini pool code.

`reserveType` has limited usefulness. I think it complicates the user experience for very little.

## [I-03] No permit on AERC6909

## [I-04] ERC6909 does not implement the rewarder

## [I-05] Some of the natspecs are not up to date, especially for the minipool logic

## [I-06] Remove all `console.log` before production

## [I-07] Remove all useless commented code

### [I-08] Variable `availableMLPLiquidity` always 0 and it is unused

### [I-08] Some errors are not properly defined and need a description. (ex: `revert()`)

### [I-09] Some zero input checks may be missing (ex: `LendingPoolAddressesProvider.setAddress()`)

---

# Conclusion

Considering the high number of critical/high vulnerabilities found in the minipool code (especially in the miniPoolBorrow logic), this report's conclusion questions whether the minipool is really necessary and needed.

## Minipool pros/cons

Here is what could be done to make cod3x lending pareto compatible (easier, safer, more modular).

- We keep only the main lending pool with upgraded prama and rehypothecation
- We delete the minipool (yeah, that's sad), so we keep the already secure AAVE codebase.
- Now we will have a unique pool design (the main lending pool) that has rehypothecation + rewarder (this is not the case for minipool).
- We build a wrapper for aToken and a seamless experience for "tranched" asset borrowers (to receive the underlying directly instead of the aToken).
- So any lending pool can accept aToken from any other lending pool.
- We build a convenient UX for deploying new pools, so we become the morpho of aave type of lending pool.
- And that's it; with minimal effort, we have something that does the job perfectly.

Yes, we are removing the miniPool borrow, but I don't think this is a killer feature. I think 99% of the time, aTokens deposited by users will be enough to cover the borrowers needs.

`

| | Remove minipool | Keep minipool |
|---|---|---|
| Simplicity | ✅✅✅✅ **(only one logic : the main lending pool)** | ✅ **(2 similar logics)** |
| Security | ✅✅✅✅ | ❌❌❌ **(most critical bugs reported come from the minipool borrow)** |
| Frontend implementation | ✅✅✅✅✅ **(super easy)** | ✅ **(frontend team will need to build a framework for ERC6909)** |
| onchain management | ✅✅ | ✅✅✅✅ **(ERC6909 may simplify onchain management)** |
| Modularity | ✅✅✅✅✅ **(only one design interchangeable)** | ✅✅✅ **(minipools will need a main lending pool to be deployed)** |
| Audit cost | ✅ **(audit size divided by 2)** | ❌ |
| Minipool borrow capacity | ❌ | ✅ |
| Rehypothecation capacity on all markets | ✅ | ❌ **(to be developped on minipool)** |

| | Remove minipool | Keep minipool |
|---|---|---|
| Rewarder on all markets | ✅ | ❌ **(to be developped on minipool)** |
| Different aToken and debtToken implementations possible on different assets | ✅ | ❌ **(the entire aERC6909 will need to be modify)** |