# Exactly Finance Fixed Lending Protocol Smart Contracts Review

By: ChainSafe Systems

May, 2022

# Exactly Finance Fixed Lending Protocol Smart Contracts Review

Auditors: Oleksii Matiiasevych, Tanya Bushenyova, Anderson Lee

## Warranty

## 1. Introduction

Exactly Finance requested ChainSafe Systems to perform a review of the Fixed Lending Protocol smart contracts. The contracts can be identified by the following git commit hash:

`aff440306ca65335aaec666158da43697d6b6a61`

There are 11 contracts, interfaces and libraries in scope.

After the initial review, Exactly Finance team applied a number of updates which can be identified by the following git commit hash:

`cc68303704cf9a3aaad0bfc49968033acc42a63b`

Additional verification was performed after that.

## 2. Disclaimer

The review makes no statements or warranties about the utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about the fitness of the contracts for any specific purpose, or their bug free status.

## 3. Executive Summary

All the initially identified high severity issues were fixed and are not present in the final version of the contract.

There are **no** known compiler bugs for the specified compiler version (0.8.14), that might affect the contracts' logic.

There were 2 critical, 0 major, 10 minor, 59 informational/optimizational issues identified in the initial version of the contracts. The critical issues found in the contracts were not present in the final version of the contracts, though new minor issues were found. They are described below for historical purposes. A number of substantial logic changes were introduced to the code which is out of scope of the verification step. Those include `FixedLender.liquidate()` , `InterestRateModel.getRateToBorrow()` , and `PoolAccounting.updateSmartPoolAssetsAverage()` functions along with a `Previewer` contract.

We are looking forward to future engagements with the Exactly team.

## 4. Critical Bugs and Vulnerabilities

Two **critical** issues (5.17, 5.19) were identified in the contracts that would allow a malicious actor to borrow all the deposited assets from the protocol without leaving anything for collateral.

## 5. Line By Line Review. Fixed Issues

5.1. PoolLib, line 16, 23. Note, the description of the `MaturityPool` struct doesn't correspond to the code: `earnings` and `earningsSP` are described but not present in the code, and `earningsUnassigned` is present in the code but not described.

5.2. PoolLib, line 43. Note, the return param `smartPoolDebtReduction` is not described.

5.3. PoolLib, line 54. Note, the return param `smartPoolDebtReduction` is not described. Also, "an operation to repay money to maturity pool" would be a more correct description.

5.4. PoolLib, line 112. Optimization, in the `withdrawMoney()` function there is no need to introduce `newSupplySP` local variable because it is used only once.

5.5. PoolLib, line 120. Note, the `currentTimeStamp` and `earningsSP` are not described.

5.6. PoolLib, line 141. Optimization, in the `accrueEarnings()` function, the `secondsTotalToMaturity==0` expression is always false due to the check on line 130.

5.7. PoolLib, line 169. Note, description of return params is missing.

5.8. PoolLib, line 177. Note, in the `distributeEarningsAccordingly()` function return variable names `(earningsA, earningsB)` are confusing.

5.9. PoolLib, line 182. Note, description of the return param is missing.

5.10. PoolLib, line 204. Note, description of the return param is missing.

5.11. PoolLib, line 230. Note, irrelevant comment. Should be about checking maturity.

5.12. FixedLender, line 49. Note, the `caller` param is not described.

5.13. FixedLender, line 54. Note, the `WithdrawAtMaturity` event misses comments on some parameters.

5.14. FixedLender, line 63. Note, the `receiver` param is not described.

5.15. FixedLender, line 69. Note, the `BorrowAtMaturity` event misses comments on some parameters.

5.16. FixedLender, line 119. Note, the `MaxFuturePoolsUpdated` event comment is outdated, max future pools is represented with 0 decimals.

5.17. FixedLender, line 172. **Critical,** the `beforeWithdraw()` hook validates the `msg.sender` shortfall instead of the shares `owner`.

5.18. FixedLender, line 177. Optimization, the `smartPoolBalance` variable is read twice from storage.

5.19. FixedLender, line 207. **Critical**, the `transferFrom()` function validates the `msg.sender` shortfall instead of the shares `owner`.

5.20. FixedLender, line 253. Note, in the `liquidate()` function, the `maxAssetsAllowed` parameter description is missing in the comments of the `liquidate()` function.

5.21. FixedLender, line 324. Optimization, in the `borrowAtMaturity()` function, the `smartPoolBalance` variable is read twice from storage.

5.22. FixedLender, line 332. Note, the `receiver` and `maturityAssets` params are not described.

5.23. FixedLender, line 355. Note, the `receiver`, `owner` and `assetsDiscounted` params are not described.

5.24. FixedLender, line 382. Optimization, the `smartPoolBalance` variable is read twice from storage.

5.25. FixedLender, line 389. Note, the `maxAssetsAllowed` param is not described.

5.26. FixedLender, line 415. Note, the `maxAssetsAllowed` param is not described.

5.27. FixedLenderETHRouter, line 54. Note, the `withdrawAtMaturityETH()` function could use `unwrapAndTransfer()` in the end.

5.28. PoolAccounting, line 30. Note, the `RepayVars.penalties` variable is not used.

5.29. PoolAccounting, line 33. Note, the `RepayVars.amountStillBorrowed` variable is not used.

5.30. PoolAccounting, line 116. Optimization, in the `borrowMP()` function the `earningsSP` variable is increased instead of being assigned.

5.31. PoolAccounting, line 131. Optimization, the `smartPoolBorrowed` variable is read from storage 3 times in the `borrowMP()` function.

5.32. PoolAccounting, line 159. Note, the “It doesn’t transfer or.” comment looks unfinished.

5.33. PoolAccounting, line 159. Note, description of the return param `earningsSP` is missing.

5.34. PoolAccounting, line 192. Note, the `minAmountRequired` and return params are not described.

5.35. PoolAccounting, line 252. Note, the `maxAmountAllowed` param is not described.

5.36. PoolAccounting, line 288. Note, the `“Math.min”` comment is not relevant.

5.37. PoolAccounting, line 317. Optimization, in the `repayMP()` function, the `(repayVars.scaleDebtCovered.principal + repayVars.scaleDebtCovered.fee)` expression could be replaced with `debtCovered.`

## 6. Line By Line Verification. Remaining and Acknowledged Issues.

6.1. PoolLib, line 67. Note, the `maxDebt` param is not described.

6.2. PoolLib, line 93. Note, the `maxDebt` param is not described.

6.3. PoolLib, line 148. Note, the “Needs for the amount to be less than the principal and the fee” part is not mandatory for the `scaleProportionally()` function.

6.4. PoolLib, line 233. Note, the `hasMaturity()` function is not used.

6.5. TSUtils, line 32. Note, the `getPoolState()` function will return `State.MATURED` for `poolID==0.`

6.6. Auditor, line 96. **Minor**, the `liquidationIncentive` variable is not validated in the constructor and could be an invalid value.

6.7. Auditor, line 109. **Minor**, the `enterMarkets()` function will exit if some market has already entered. Consider skipping such a market instead.

6.8. Auditor, line 179. **Minor**, the `collateralFactor` of the market could be set to an invalid value in the `enableMarket()` function.

6.9. Auditor, line 244. Optimization, the invariant assert in the `validateBorrowMP` is impossible to break due to a statement on line 240.

6.10. Auditor, line 251. **Minor**, the `validateBorrowMP()` function restricts borrowing that reaches the `borrowCap`, instead it should only restrict if the cap is surpassed.

6.11. Auditor, line 331. Note, the `getMarketData()` function includes the input parameter `fixedLender` in the return struct.

6.12. Auditor, line 401. Optimization, the `accountAssets[account]` variable is read from storage 2 times in case of `validateAccountShortfall()` function execution. Consider passing it as a parameter.

6.13. Auditor, line 403. Note, the `accountLiquidity()` function potentially performs `allMarkets.legnth*224` operations. If the execution of this function will take more than block gas limit, then users could restrict themselves from being liquidated.

6.14. FixedLender, line 22. Optimization, the `assetSymbol` variable is only used for price retrieval from the oracle. Contract address itself could be used instead, or at least `bytes32` instead of a string.

6.15. FixedLender, line 26. Optimization, the `accumalatedEarningsSmoothFactor` variable could fit into a smaller `uint` type

6.16. FixedLender, line 27. Optimization, the `lastAccumulatedEarningsAccrual ` the variable could fit into `unit32` and placed together with `maxFuturePools.`

6.17. FixedLender, line 139. **Minor**, the `maxFuturePools` value is not validated in the constructor and could be set to 0.

6.18. FixedLender, line 140. **Minor**, the `accumulatedEarningsSmoothFactor` value is not validated in the constructor and could be set above `4e18`.

6.19. FixedLender, line 143. Note, the `totalAssets()` function is excessively optimized. As this function is only used by external observers, consider improving readability over efficiency.

6.20. FixedLender, line 176. Optimization, in the `beforeWithdraw()` function, the `smartPoolEarningsAccumulator` variable is read from storage 2 times. One in the function itself and another inside `smartPoolAccumulatedEarnings().`

6.21. FixedLender, line 185. Optimization, in the `afterDeposit()` function, the `smartPoolEarningsAccumulator` variable is read from storage 2 times. One in the function itself and another inside `smartPoolAccumulatedEarnings().`

6.22. FixedLender, line 216. **Minor**, in the `setMaxFuturePools()` function, the new `maxFuturePools` value should be limited to 224.

6.23. FixedLender, line 243. Note, in the `liquidate()` function, the return param `repaidAssets` is not described.

6.24. FixedLender, line 299. Note, the `receiver` and `borrower` params are not described. Return param is not described.

6.25. FixedLender, line 305. Note, the `borrowAtMaturity()` function will work even with 0 amount.

6.26. FixedLender, line 338. Note, the `depositAtMaturity()` function will work even with 0 amount.

6.27. InterestRateModel, line 50. **Minor**, the `spFeeRate` value is not validated in the constructor and could be set above 20%.

6.28. PoolAccounting, line 41. Optimization, the `penaltyRate` and `smartPoolReserveFactor` variables could be packed together into a one 32-byte storage slot by using smaller `uint` types for them.

6.29. PoolAccounting, line 69. **Minor**, the `penaltyRate` value is not validated in the constructor and could be set to disallowed value.

6.30. PoolAccounting, line 70. Minor, the `smartPoolReserveFactor` value is not validated in the constructor and could be set above 20%.

6.31. PoolAccounting, line 236. Optimization, the `smartPoolBorrowed` storage variable is read 2 times in the `withdrawMP()` function.

6.32. PoolAccounting, line 243. Note, in the `withdrawMP()` function, `PoolLib.distributeEarningsAccordingly()` could be called with the withdrawn amount instead of the `redeemAmountDiscounted`.

6.33. PoolAccounting, line 301. Optimization, the `pool.earningsUnassigned` variable is read from storage 2 times in the `repayMP()` function. First time in the `pool.accrueEarnings()` function.

6.34. PoolAccounting, line 353. Optimization, the `getAccountBorrows()` function will read the `penaltyRate` variable from storage for every maturity.

# 7. Line By Line Verification. New Issues.

7.1. FixedLender, line 156. Note, `memMaxFuturePools` is used only once in the function, no need to create a local variable for it.

7.2. FixedLender, line 500. Note, the `debtCovered` return param is not described.

7.3. PoolAccounting, line 78. Minor, the `dampSpeedUp` value is not validated in the `constructor` and could be set to disallowed value.

7.4. PoolAccounting, line 79. Minor, the `dampSpeedDown` value is not validated in the `constructor` and could be set to disallowed value.

ChainSafe

[illegible]

Oleksii Matiiasevych

[illegible]

Tanya Bushenyova

[illegible]

Anderson Lee