Exactly Protocol Update Smart Contracts Review

By: ChainSafe Systems

October 2022

Exactly Protocol Update Smart Contracts Review

Auditors: Tanya Bushenyova, Anderson Lee, Oleksii Matiiasevych

WARRANTY

This Code Review is provided on an “as is” basis, without warranty of any kind, express or implied. It is not intended to provide legal advice, and any information, assessments, summaries, or recommendations are provided only for convenience (each, and collectively a “recommendation”). Recommendations are not intended to be comprehensive or applicable in all situations. ChainSafe Systems does not guarantee that the Code Review will identify all instances of security vulnerabilities or other related issues.

Introduction

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

`af5b4907ec63fe034fb8dde62bae99222d8407bc`

There are 7 contracts in scope including their parent contracts and interfaces.
After the initial review, Exactly Protocol team applied a number of updates which can be identified by the following git commit hash:

`66209d6c44a6180bb7fbdc0bc4f46f292d2315a4`

Additional verification was performed after that. The verification focused solely on the previous findings and did not cover the new logic introduced together with the fixes.

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.

Executive Summary

There are no known compiler bugs for the specified compiler version (0.8.17), that might affect the contracts' logic.

There were 0 critical, 0 major, 3 minor, 25 informational/optimizational issues identified in the initial version of the contracts. The code base has gained in complexity since the previous version along with introducing the upgradeability for the core Market and Auditor contracts. The admin functions should be called with extra care to make sure the parameters are correct, as there are no validations left. The minor issues found in the contracts were not present in the final version of the contracts. They are described below for historical purposes.

Critical Bugs and Vulnerabilities

No critical issues were identified.

Line by Line Review. Fixed Issues

1. Market, line 244. Optimization, the `borrowAtMaturity()` function reads `floatingAssetsAverage` from storage multiple times.

2. Market, line 253. Minor, the `borrowAtMaturity()` function reads an outdated `floatingDebt` value. It should call `updateFloatingDebt()` to make sure it uses the latest value.

3. Market, line 272. Optimization, the `borrowAtMaturity()` function could pointlessly update `pool.unassignedEarnings` if `newUnassignedEarnings` is 0.

4. Market, line 337. Optimization, the `floatingBackupBorrowed` is read twice from storage in the `withdrawAtMaturity()` function.

5. Market, line 340. Minor, the `withdrawAtMaturity()` function reads an outdated `floatingDebt` value. It should call `updateFloatingDebt()` to make sure it uses the latest value.

6. Market, line 429. Note, the `noTransferRepayAtMaturity()` function has an outdated variable name in the comments. It should state `actualRepayAssets` instead of `repayAmount`.

7. Market, line 444. Note, the `noTransferRepayAtMaturity()` function has an outdated comment about `‘SP debt’` which should be talking about floating borrowed instead.

8. Market, line 565. Optimization, the `clearBadDebt()` function calls `spreadBadDebt()` multiple times, every time updating storage. Consider accumulating the `badDebt` sum and call the spread only once.

9. Market, line 818. Note, in the `updateFloatingDebt()` function the `interestRateModel` is read only once, no need to store it in a local var.

10. Market, line 820. Optimization, the `floatingAssets` is read twice from storage in the `updateFloatingDebt()` function.

11. Market, line 830. Optimization, the `updatefloatingDebt()` function updates `floatingAssets` twice. Second time in the `chargeTreasuryFee()` function.

12. Market, line 857. Note, in the `totalFloatingBorrowAssets()` function the `interestRateModel` is read only once, no need to store it in a local var.

13. Market, line 869. Minor, the `totalAssets()` function could incorrectly calculate the `latestMaturity` `backupEarnings` if the `lastAccrual < maturity`. Like now = 100, maturity = 80, lastAccrual = 70.

14. Market, line 1060. Note, the `WithdrawAtMaturity` event comments are not correct, `assetsDiscounted` is the amount withdrawn.

15. Market, line 1189. Note, the `AlreadyInitialized()` error is not used.

16. Auditor, line 148. Optimization, the `checkBorrow()` function could pass `Market(0)` to `accountLiquidity()` call.

17. Auditor, line 234. Note, the `checkLiquidation()` function uses an in-place long constant. Consider introducing a contract level constant `ASSETS_THRESHOLD = type(uint256).max / 1e18`;

18. Auditor, line 335. Note, the `enableMarket()` function sets the `adjustFactor` twice with the same value.

19. MarketETHRouter, line 21. Optimization, in the `unwrap()` modifier the second parameter is always `msg.sender` and can be omitted.

20. MarketETHRouter, line 98. Optimization, in the `unwrapAndTransfer()` modifier the second parameter is always `msg.sender` and can be omitted.

Line by Line Review. Acknowledged Findings.

1. Market, line 115. Optimization, the `floatingDebt` is read from storage multiple times in the `borrow()` function that calls `updateFloatingDebt()`.

2. Market, line 170. Optimization, the `floatingDebt` is read from storage multiple times in the `noTransferRefund()` function that calls `updateFloatingDebt()`.

3. Market, line 252. Optimization, the `floatingAssets` is read from storage multiple times in the `borrowAtMaturity()` function.

4. Market, line 340. Optimization, the `floatingAssets` is read from storage twice in the `withdrawAtMaturity()` function.

5. Market, line 643. Optimization, the `floatingAssets` is read from storage thrice in the `beforeWithdraw()` function.

6. Market, line 655. Optimization, the `floatingAssets` is read from storage thrice in the `afterDeposit()` function.

7. Auditor, line 162. Optimization, the `accountMarkets[account]` variable is read from storage twice in the `checkShortfall()` function.

8. FixedLib, line 223. Note, the `getPoolState()` function will return `State.MATURED` for `poolId == 0`.

[illegible] [illegible] [illegible]

Tanya Bushenyova Anderson Lee Oleksii Matiiasevych