Hyperdrive Smart Contracts Review

By: ChainSafe Systems

June 2023

Hyperdrive Smart Contracts Review

Auditors: Tanya Bushenyova, Anderson Lee, Oleksii Matiiasevych

WARRANTY

Introduction

Delv requested ChainSafe Systems to perform a review of the Hyperdrive contracts implementing their protocol for trading assets that can be redeemed for their full face value at maturity. The contracts can be identified by the following git commit hash:

`9e960c556654225345ddaad1ce81c81871e218d1`

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

`1bcf5fe45b9d3dd02741302dd639104338e79c21`

Additional verification was performed after that.

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 (0.8.19), that might affect the contracts' logic.

There were 0 critical, 0 major, 1 minor, 28 informational/optimizational issues identified in the initial version of the contracts. All the minor and most of the informational/optimizational issues found in the contracts were not present in the final verified version of the contracts. They are described below for historical purposes. We enjoyed working with the Delv team, and liked how engaged they were in the discussion and improvement process throughout the review and how they tracked and implemented the recommendations of the audit.

Critical Bugs and Vulnerabilities

No critical issues were identified.

Line by Line Review. Fixed Issues

1. HyperdriveFactory.sol, line 52: Note, the `updateImplementation()`, `updateGovernance()`, `updateHyperdriveGovernance()`, `updateFees()` functions have the same code for checking `msg.sender`. Consider adding an internal function or a modifier.

2. HyperdriveFactory.sol, line 65: Note, wrong comments.

3. HyperdriveFactory.sol, line 74: Note, wrong comments.

4. HyperdriveFactory.sol, line 83: Note, wrong comments.

5. DSRHyperdriveFactory.sol, line 36: Note, wrong comment: the comment mentions “aave hyperdrive instance” and it should mention DSR.

6. AaveHyperdrive.sol, line 108: Note, it’s not clear why it’s necessary to proceed with the withdrawal in the `withdraw()` function if `withdrawValue == 0` and not to revert in this case. Removing this condition or adding a comment might be helpful.

7. AaveHyperdrive.sol, line 125: Minor, `sharePrice` is calculated incorrectly in the `_withdraw()` function, it should be the opposite (`withdrawValue / shares`).

8. AaveHyperdrive.sol, line 138: Optimization, `totalShares` is read multiple times from storage in the `_pricePerShare()` function.

9. DsrHyperdrive.sol, line 58: Note, wrong comment in the `_deposit()` function. If `asUnderlying` is false, the transaction is reverted.

10. DsrHyperdrive.sol, line 100: Note, the `asUnderlying` parameter of the `_withdraw()` function is not described.

11. DsrHyperdrive.sol, line 115: Note, a check that the `shares` amount doesn’t exceed `totalShares` could be added in the `_withdraw()` function (like in AaveHyperdrive contract).

12. DsrHyperdrive.sol, line 155: Optimization, `pot` is read multiple times from storage in the `chi()` function.

13. AaveHyperdriveDataProvider.sol, line 29: Note, there is no getter for `totalShares`.

14. ERC20Permit.sol, line 47: Note, wrong comment. The comment “By setting these addresses to 0” is incorrect: the balances of these addresses are set to `type(uint256).max`.

15. ERC20Permit.sol, line 207: Optimization, `nonces[owner]` is read multiple times from storage in the `permit()` function.

16. BondWrapper.sol, line 17: Note, a check in the constructor could be added to check that the `mintPercent` is less than 10000 (100%).

17. BondWrapper.sol, line 57: Optimization, checking `maturityTime` could be performed at the first line of the `mint()` function before creating `assetId`.

18. HyperdriveDataProvider.sol, line 155. Note, the `query()` function is expected to revert with a success output as part of a force-revert delegatecall pattern. Doing an unsuccessful revert could have undefined behavior. Consider using a staticcall wrapped delegate call pattern, or make sure there are never unexpected reverts in DataProviders.

19. HyperdriveLP.sol, line 305. Note, the `removeLiquidity()` function has `overestimatedProceeds` calculation which would be more readable as `startingPresentValue.mulDivDown(uint256(-withdrawalShares), lpTotalSupply`), because in this way the coefficient part will have the same units.

20. HyperdriveLP.sol, line 483. Note, the `_compensateWithdrawalPool()` function has `maxSharesReleased` calculation which would be more readable as `_lpTotalSupply.mulDivDown(_withdrawalProceeds, _presentValue)`, because in this way the coefficient part will have the same units.

21. HyperdriveTWAP.sol, line 33. Optimization, the `recordPrice()` function excessively reads the `_buffer[head].timestamp` which is equal to already loaded `_oracle.lastTimestamp`.

Line by Line Review. Acknowledged Findings.

1. AaveHyperdrive.sol, line 79: Optimization, `totalShares` is read multiple times from storage in the `_deposit()` function.

2. AaveHyperdrive.sol, line 106: Optimization, `totalShares` is read multiple times from storage in the `_withdraw()` function.

3. DsrHyperdrive.sol, line 86: Optimization, `totalShares` is read multiple times from storage in the `_deposit()` function.

4. DsrHyperdrive.sol, line 115: Optimization, `totalShares` is read multiple times from storage in the `_withdraw()` function.

5. DsrHyperdriveDataProvider.sol, line 73: Note, the `_pricePerShare()` function code is repeated in DsrHyperdrive and DsrHyperdriveDataProvider contracts. Consider including this code once (rewriting the contracts or putting the duplicated code in a library).

6. BondWrapper.sol, line 110: Note, `mintedFromBonds` could be calculated earlier and passed as `_minOutput` to the `hyperdrive.closeLong()` function.

7. BondWrapper.sol, line 119: Note, `mintedFromBonds` could be subtracted from `receivedAmount` only `if (!andBurn)` in the `close()` function. In this case, double calculation would not be necessary (subtracting and then adding `mintedFromBonds`).

8. HyperdriveTWAP.sol, line 34. Optimization, the `recordPrice()` function could omit reading the `_buffer[head]` at all, if all the values from it would be duplicated into the `_oracle.`

Anderson Lee

Tanya Bushenyova

Oleksii Matiiasevych