Grappa Options Protocol Smart Contracts Review

By: ChainSafe Systems

February 2023

Sponsored by Hashnote Finance

Grappa Options Protocol Smart Contracts Review

Auditors: Anderson Lee, Tanya Bushenyova, Oleksii Matiiasevych

WARRANTY

Introduction

Hashnote Finance requested ChainSafe Systems to perform a review of the Grappa contracts implementing their options protocol ecosystem core. The contracts can be identified by the following git commit hashes:

`88e9cd5c6d51c736fb1df686a04d334b1594fdc7`
`2be6901d4ebeaf3996aa38d7cde6c75c9f6bde0d`

Everything except for the non-cross-margin engines.
After the initial review, Hashnote team applied a number of updates which can be identified by the following git commit hashes

`ad89bbfef709184fc27bca6b72fcbe27024f004a`

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

There were 0 critical, 1 major, 4 minor, 30 informational/optimizational issues identified in the initial version of the contracts. All critical and major 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 Hashnote Finance team, and liked how engaged they were in the discussion and improvement process throughout the review.

Critical Bugs and Vulnerabilities

No critical issues were identified.

Line by Line Review. Fixed Issues

`Commit Hash: 88e9cd5c6d51c736fb1df686a04d334b1594fdc7`

1. Grappa, line 54. Note, the `nextAssetId` name is confusing, it should be the `lastAssetId` instead.

2. Grappa, line 58. Note, the `nextengineId` name is confusing, it should be the `lastEngineId` instead.

3. Grappa, line 62. Note, the `nextOracleId` name is confusing, it should be the `lastOracleId` instead.

4. AccountUtil, line 113. Optimization, the `sum()` function doesn't make sense because amounts of different tokens are mixed together. It is only used to check if the `PositionOptim[]` is empty so consider introducing an `isEmpty()` function instead, which would return false immediately upon finding a positive amount. It will make it even more optimized to work with a storage pointer instead of memory.

5. ChainlinkOracle, line 158. ***Minor***, the `_toPriceWithUnitDecimals()` function does a double division which results in an excessive precision loss. Instead of doing A / B / C, consider doing A / (B * C).

6. ChainlinkOracle, line 201. ***Minor***, the `_getLastPriceBeforeExpiry()` function must make sure that the `nextRoundUpdatedAt` is ***strictly greater*** than `_expiry`, currently it also allows to equal it.

7. ActionUtil, line 169. Optimization, the `createSettleAction()` function excessively encodes data as 32 empty bytes. Consider encoding it as an `empty` bytes array.

8. ArrayUtil, line 284. ***Major***, the `quickSort()` function with indexes mixes up order of indexes for equal elements. For the array `[1,1,1,1]` the expected `indexArray` should be [0,1,2,3], but instead it is `[2,3,0,1]`.

9. BalanceUtil, line 53. Optimization, the `sum()` function doesn't make sense because amounts of different collaterals are mixed together. It is only used to check if the `Balance[]` is empty so consider introducing `isEmpty()` function instead, which would return false immediately upon finding a positive amount. It will make it even more optimized to work with a storage pointer instead of memory.

`Commit Hash: 2be6901d4ebeaf3996aa38d7cde6c75c9f6bde0d`

10. Grappa, line 16. Note, the `IGrappa` import is not used.

11. Grappa, line 57. Note, the comment mentions `tokenId` instead of `engineId`.

12. Grappa, line 61. Note, the comment mentions `tokenId` instead of `oracleId`.

13. OptionToken, line 12. Note, the `TokenIdUtil` import is not used.

14. OptionToken, line 13. Note, the `ProductIdUtil` import is not used.

15. CrossMarginEngine, line 14. Note, the `IOracle` import is not used.

16. CrossMarginEngine, line 22. Note, the `ArrayUtil` import is not used.

17. BaseEngine, line 14. Note, the `IERC20` interface is imported twice.

18. BaseEngine, line 39. Note, the comment “allowedExecutionLeft refers to the time left the grantee can update the sub-accounts.” is misleading. The `allowedExecutionLeft` variable is a counter of updates but the comment implies that it refers to a timestamp.

19. BaseEngine, line 243. Note, the `_addOption()` function comment mentions “token being burn” while it should mention the token being added.

20. BaseEngine, line 270. Note, the `_removeOption()` function comment mentions “transfer the option token in” while it should mention that the token is being transferred out of the contract.

21. ChainlinkOracleDisputable, line 4. Note, the `FixedPointMathLib` import is not used.

Line by Line Review. Rejected Findings, with reasons.

1. ChainlinkOracle, line 119. ***Power***, the `setAggregator()` function lets the owner control the price. ***Reason for rejection***: the function will be controlled by a multisig.

2. ChainlinkOracleDisputable, line 51. ***Power***, the `disputePrice()` function lets the owner control the price. ***Reason for rejection***: the function will be controlled by a multisig.

Line by Line Review. Acknowledged Findings.

1. Grappa, line 334. Note, in multiple places in the code `Utils` are used as a function call or as a self call interchangeably. Consider using the same style everywhere.

2. Grappa, line 499. Optimization, the `_addToPayouts()` function appends the new collaterals to the end of the list, making subsequent search more and more expensive. Consider appending to the beginning of the list instead. Another option would be to implement `indexOf()` starting from the end of the list.

3. BaseEngine, line 119. ***Minor***, the `onERC1155Received()` function should only accept `optionToken` transfers.

4. BaseEngine, line 127. ***Minor***, the `onERC1155BatchReceived()` function should only accept `optionToken` transfers.

5. BaseEngine, line 305. Optimization, the `_transferShort()` function could be redesigned to avoid additional `aboveWater` check by transferring into the caller's `subAccount`.

6. AccountUtil, line 148. Optimization, the `removePositionAt()` function could do an excessive reassignment in case of last element removal.

7. CrossMarginLib, line 45. Optimization, the `addCollateral()` function searches the storage which will get expensive quickly. Consider maintaining a bit map of collaterals instead.

8. ActionUtil, line 11. Optimization, the `create*()` functions encode data with ABI encoding which pads all the arguments to 32 bytes with zeroes. Consider using `encodePacked()` instead together with custom decoding routines.

9. ActionUtil, line 192. Optimization, the `append(ActionArgs[], ...)` function introduces a memory leak by wasting previously allocated space. Consider using a linked list or alike structure instead.

10. ActionUtil, line 204. Optimization, the `append(BatchExecute[], ...)` function introduces a memory leak by wasting previously allocated space. Consider using a linked list or alike structure instead.

11. BalanceUtil, line 10. Optimization, the `append()` function introduces a memory leak by wasting previously allocated space. Consider using a linked list or alike structure instead.

12. BalanceUtil, line 49. Optimization, the `remove()` function could do an excessive reassignment in case of `last` element removal.

Anderson Lee

Tanya Bushenyova

Oleksii Matiiasevych