Hashnote Vaults And Auction Smart Contracts Review

By: ChainSafe Systems

Febraury 2023

Hashnote Vaults And Auction Smart Contracts Review

Auditors: Anderson Lee, Tanya Bushenyova, Oleksii Matiiasevych

WARRANTY

Introduction

Hashnote Finance requested ChainSafe Systems to perform a review of the vaults and auction contracts used with their options protocol ecosystem. The contracts can be identified by the following git commit hashes:

```
029f7f43c708b4b9a36000196f95e1fcc332325b
e6d1dbbd7e109442bf3ce9015b49a75e6fcd6687
```

Everything in scope that is present in the second commit.
After the initial review, Hashnote team applied a number of updates which can be identified by the following git commit hashes:

```
48ce04a3a43ec1dfbb335b6d4dd06aba3160c02f
```

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 1 critical, 4 major, 2 minor, 40 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: 029f7f43c708b4b9a36000196f95e1fcc332325b
```

1. IBatchAuction, line 32. Note, the `Auction.availableSize` comment is misleading. The `availableSize` value is updated only during settlement.

2. BatchAuctionQ, line 64. Note, the `computeFills()` function for-loop condition `(i >= 0)` is always true.

3. BatchAuctionQ, line 74. ***Critical***, the `computeFills()` function could revert forever when `accessing bidOrder[i]` with `i` being underflowed if the first bid was removed.

4. BatchAuction, line 24. Note, the `UNIT_DECIMALS` constant is not used.

5. BatchAuction, line 58. Note, the `createAuction()` function allows a zero `minBidSize`.

6. BatchAuction, line 59. Note, the `createAuction()` function allows a zero `totalSize`.

7. BatchAuction, line 70. Optimization, the `createAuction()` function reads `auctionsCounter` variable from storage twice.

8. BatchAuction, line 157. Note, the `removeBid()` function has a misleading error `BA_Unauthorized` for invalid `bidId`.

9. BatchAuction, line 183. Note, the `settleAuction()` function could be allowed to execute at `auction.endTime`, because other actions are already not allowed.

10. BatchAuction, line 185. ***Major***, the `settleAuction()` function could be left unexecuted and keep bidders' money locked forever if the seller decides so.

11. BatchAuction, line 204. ***Major***, the `settleAuction()` function could be left unexecuted and keep the bidders money locked forever if the `clearingPrice` is negative and seller removes allowance on `biddingToken`.

12. BatchAuction, line 226. Optimization, the `claim()` function reads `queue.bidOwnerList.length` from storage multiple times.

`Commit Hash: e6d1dbbd7e109442bf3ce9015b49a75e6fcd6687`

13. FeeUtil, line 99. Note, the `processFees()` function has a misleading comment about `currentBalance` at round 1 being 0. This is not true if the pendingAmount > 0, because when `pendingAmount` grows, the `currentBalance` grows as well.

14. StructureUtil, line 50. Note, the `stageStructure()` function excessively validates the `strikes` and `instruments` lengths, even though it is also validated in the calling function.

15. StructureUtil, line 185. Note, in the `_createMarginDepositActions()` function it would be semantically correct to calculate the amount as `balances[i].mulDivDown(structuresToMint, maxStructures)`.

16. Vault, line 33. Note, Missing descriptions for `_manager, _oracle, _vaultPauser, _batchAuction, _collaterals, _leverageRatio, _roundConfig`.

17. VaultUtil, line 160. Note, the `verifyInitializerParams()` function restricts `collaterals` with zero `decimals`, though ERC20 standard allows that.

18. HashnoteBatchAuction, line 395. Optimization, the `auction.collaterals` value is read from storage multiple times in the `claim()` function.

19. HashnoteVaultPauser, line 113. Optimization, the `pausePosition()` function reads `pausedPosition.round` value from storage twice.

20. HashnoteVaultPauser, line 159. Major, the `withdrawCollaterals()` function does not verify if the `_destination` is whitelisted or not, it verifies `msg.sender` twice instead.

21. HashnoteVaultPauser, line 205. Note, the `recoverTokens()` function could have an `amount` argument to make it easier to recover tokens of multiple recipients.

22. HashnoteOptionsVault, line 100. Note, the `setBatchAuction()` could emit an event for easier accounting.

23. HashnoteOptionsVault, line 115. Note, the `setAuctionDuration()` could emit an event for easier accounting.

24. HashnoteOptionsVault, line 126. Note, the `setPremium()` could emit an event for easier accounting.

25. HashnoteOptionsVault, line 178. Optimization, the `stageStructure()` function excessively checks if `currentRound == 1` in the else clause where it is always `false`.

26. HashnoteOptionsVault, line 231. Optimization, the `startAuction()` function could have a second `options.length == 0` condition placed inside of the first one.

27. HashnoteOptionsVault, line 248. Optimization, the `startAuction()` function excessively reads the `auctionId` value from storage to emit an event.

28. HashnoteVault, line 5. Note, the use of `SafeMath` is not necessary since Solidity compiler version 0.8.x+ is used.

29. HashnoteVault, line 127. Note, the `baseInitialize()` function could be protected by `onlyInitializing` modifier that is available in a v4.4.1 and above version of openzeppelin lib.

30. HashnoteVault, line 181. Note, the `setManager()` could emit an event for easier accounting.

31. HashnoteVault, line 195. Note, the `setFeeRecipient()` could emit an event for easier accounting.

32. HashnoteVault, line 237. Note, the `setCap()` could emit an event for easier accounting.

33. HashnoteVault, line 250. Note, the `setVaultPauser()` could emit an event for easier accounting.

34. HashnoteVault, line 261. Note, the `setWhitelist()` could emit an event for easier accounting.

35. HashnoteVault, line 276. Note, the `setRoundConfig()` could emit an event for easier accounting.

36. HashnoteVault, line 325. Optimization, the `depositFor()` excessively declares a `newAmount` variable that is immediately used.

37. HashnoteVault, line 367. Optimization, the `quickWithdraw()` function reads `depositReceipts[msg.sender]` from storage twice. Consider reading the `amount` and `round` values into memory in a single operation.

38. HashnoteVault, line 394. Optimization, the `requestWithdraw()` function reads `depositReceipts[msg.sender]` from storage twice. Consider reading the `amount` and `unredeemedShares` values into memory in a single operation.

39. HashnoteVault, line 428. Optimization, the `depositReceipt.round value` is read multiple times from storage in the `redeem()` function.

40. HashnoteVault, line 588. Optimization, the `_transferAssets()` function reads `vaultState.round` from storage, even though its value is already available in the calling function. Consider passing the `currentRound` value instead.

Line by Line Review. Rejected Findings, with reasons.

1. HashnoteBatchAuction, line 223. Note, the `_checkBidderPermissions()` call in the `cancelBid()` function might be excessive. It is checked further in the code that the owner of the bid is `msg.sender`. Besides, if after placing a bid a user is removed from the whitelist then they will be unable to cancel the bid until the end of the auction.
Reason for rejection: the user should not be allowed to withdraw collateral if removed from the whitelist to be compliant with regulations.

2. BatchAuction, line 193. ***Major***, the `settleAuction()` function could be locked forever if too many bids are placed and settling takes more gas than the block gas limit.
Reason for rejection: the auction is permissioned and does not expect to have a high number of bids in the initial stages of the protocol. If it’s going to be a problem, it will be reconsidered in future upgrades.

Line by Line Review. Acknowledged Findings.

1. FeeUtil, line 172. Optimization, the `calculateRelativeNAV()` function could start the loop from index `1` to avoid excessive calculation of primary collateral `NAV`.

2. HashnoteVaultPauser, line 88. Note, the `setWhitelist()` could emit an event for easier accounting.

3. HashnoteVaultPauser, line 118. Optimization, the `pausePosition()` function could read `pausedPosition.shares` value excessively in case the position is empty.

4. HashnoteVault, line 146. ***Minor***, the `baseInitialize()` function doesn't validate instruments for duplicates.

5. HashnoteVault, line 154. Minor, the `baseInitialize()` function doesn't validate `collaterals` for duplicates.

[illegible]

Anderson Lee

[illegible]

Tanya Bushenyova

[illegible]

Oleksii Matiiasevych