Nereus Liquidator Contracts Code Audit and Verification by Ambisafe Inc.

November, 2022

Oleksii Matiiasevych

1. **Introduction.** Nereus Finance. requested Ambisafe to perform a code audit of the Liquidator and related contracts. The contracts in question can be identified by the following git commit hash:

   `d57d0003438fe4add0bd76d63dd48c81723d7d34`

   The scope of the audit is Liquidator and CRVTokenUSDCLiquidationStrategy contracts.

   During the initial code audit, Nereus Finance team applied a number of updates which can be identified by the following git commit hash:

   `ffeb384f986b03933eb4448c788942531504e1be`

   Additional verification was performed after that.

2. **Disclaimer.** The code audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts for any specific purpose, or their bugfree status. The code audit documentation below is for internal management discussion purposes only and should not be used or relied upon by external parties without the express written consent of Ambisafe.

3. **Executive Summary.** There are **no** known compiler bugs for the specified compiler version (0.8.0), that might affect the contracts' logic. There were 0 critical, 1 major, 2 minor, 6 informational and optimizational issues identified in the initial version of the contracts. Some of the issues were addressed while others remained acknowledged.

4. **Critical Bugs and Vulnerabilities.** No critical bugs or vulnerabilities were found.

5. Initial Line By Line Review. Fixed Findings.

5.1. Liquidator, line 145. **Minor**, the **switchLiquidationStrategy()** function could activate an empty strategy.

5.2. Liquidator, line 199. Optimization, the **liquidate()** function reads **_tokenLiquidationStrategy** struct from storage multiple times. Consider reading it into memory once instead.

5.3. Liquidator, line 257. Note, the **swap()** function could be used by a **manager** to drain the contract.

6. Verification Line By Line Review. Acknowledged Findings.

6.1. Liquidator, line 22. Note, the **MAX_INT** could be defined as **type(uint256).max** for better readability.

6.2. Liquidator, line 182. Note, the **liquidate()** function could revert with underflow if the BentoBox collateral base is smaller than **minimumBalanceInBentoBox**.

6.3. Liquidator, line 191. **Minor**, in the **liquidate()** function the **amount** is confused with **shares** in the **bentoBox.withdraw()** call. This doesn't have negative effects while the ratio of shares/underlying in the BentoBox is 1:1.

6.4. Liquidator, line 261. Note, the **swap()** function could lose the revert reason. Consider doing assembly `{ revert(add(result, 32), result) }` instead.

6.5. Multicall, line 18. Note, the **multicall()** function could lose the revert reason. Consider doing assembly `{ revert(add(result, 32), result) }` instead.

6.6. CRVTokenUSDCLiquidationStrategy, line 36. **Major**, the **applyStrategy()** function is susceptible to a sandwich attack due to the call of **remove_liquidity_one_coin()** with **minOut** specified as 0.

Oleksii Matiiasevych