Ambisafe

DeXe Platform Update Contracts Code Audit and Verification by Ambisafe Inc.

November, 2023

Oleksii Matiiasevych

1. **Introduction.** DeXe Network requested Ambisafe to perform a code audit of the DeXe Platform updated contracts. The contracts in question can be identified by the following git commit hash:

   `f2fe12eeac0c4c63ac39670912640dc91d94bda5`

   All contracts in the repo are in scope.

   After the initial code audit, DeXe Network team applied a number of updates which can be identified by the following git commit hash:

   `440b8b3534d58d16df781b402503be5a64d5d576`

   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 (0.8.9), that might affect the contracts' logic. There were 1 critical, 0 major, 1 minor, 8 informational and optimizational findings identified in the initial version of the contracts. All the findings were addressed and were not found in the final version of the code.

4. **CRITICAL BUGS AND VULNERABILITIES.** One critical issue was fixed over the course of the engagement. It would allow malicious actors to corrupt the NFT power calculations in the GovUserKeeper (5.4).

5. **INITIAL LINE BY LINE REVIEW. FIXED FINDINGS.**

5.1. GovPool, line 294. Minor, the **undelegateTreasury()** function should execute **_unlock()** function on **delegatee** instead of a **msg.sender**.

5.2. GovPool, line 362. Note, the **saveOffchainResults()** function could be frontrun to steal rewards.

5.3. ERC20Gov, line 79. Note, the **blacklist()** function should work when the contract is paused, otherwise the malicious actor could always move tokens by frontrunning the blacklist transaction.

5.4. ERC721Power, line 152. **Critical**, the **recalculateNftPower()** function incorrectly adjusts the **totalPower** in case the specified **tokenId** is not minted. After the **powerCalcStartTimestamp** is passed, anyone could call **recalculateNftPower(notMintedTokenId)** to reduce the total power, which in turn will corrupt the NFTs power calculation in **GovUserKeeper**.

5.5. AbstractERC721Multiplier, line 93. Note, the **_mint()** function does not validate the **duration** value, which could lead to an overflow revert in the **_getCurrentMultiplier()** function when it is added to **mintedAt**.

5.6. DistributionProposal, line 63. Note, the **execute()** function does not validate the incoming ETH amount if the token is ETH.

5.7. DistributionProposal, line 65. Note, the **execute()** function does not validate the incoming ETH is zero if the token is not ETH.

5.8. GovUserKeeper, line 356. Optimization, the **updateMaxTokenLockedAmount()** function should return early if there is nothing to update, instead of just breaking the loop.

5.9. GovPoolUnlock, line 53. Note, the **unlockInProposals()** function assumes it is possible for **maxLockedAmount** to be less than **maxUnlocked** amount, but that should be impossible.

5.10. TokenSaleProposalBuy, line 154. Optimization, the **canParticipate()** function conditions could be optimized by moving the **_canParticipate &&** from reassignment to the **if** condition.

[illegible]

Oleksii Matiiasevych