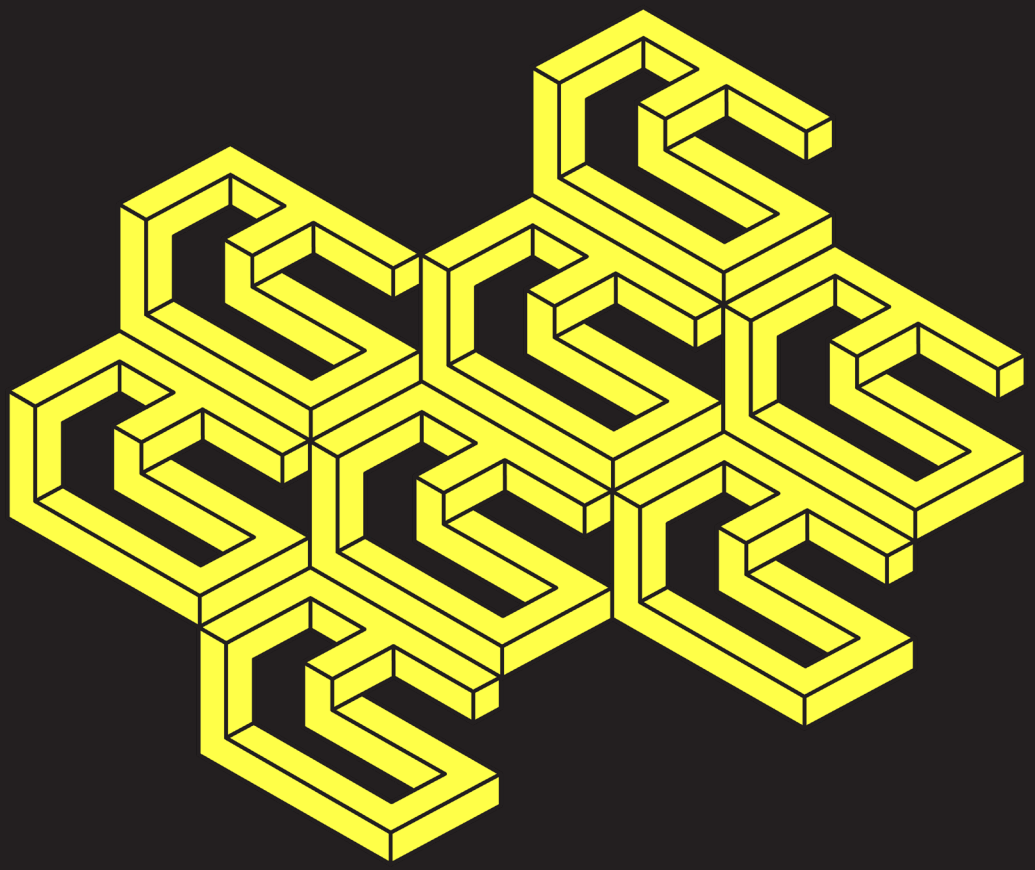

EPNS Smart Contracts Review

By ChainSafe Systems

March 2021

ChainSafe

EPNS Smart Contracts Review

Auditor: Oleksii Matiiasevych

Warranty

Executive Summary

There were 0 critical, 1 major, 2 minor, 53 informational/optimizational issues identified in this version of the contracts. There are **no** known compiler bugs, for the specified compiler version (0.6.11), that might affect the contracts’ logic.

Considerable space is available for gas efficiency improvements. In particular, Staking implementation could borrow ideas from the StakingRewards contract by Synthetix. Solidity compiler’s optimizer settings could be adjusted to account for expected number of contract executions (`runs` parameter) instead of using a default number 200. I enjoyed working with the EPNS team, especially due to their responsiveness for any inquiries during the whole engagement.

Update Verification Summary

There were 0 critical, 0 major, 0 minor, 20 informational/optimizational findings were acknowledged and left unchanged in the updated version of the contracts. There are **no** known compiler bugs, for the specified compiler version (0.6.11), that might affect the contracts’ logic. All the significant issues were addressed in this update, and 0 new issues were found.

1. Introduction

EPNS requested ChainSafe Systems to perform a review of the PUSH Tokens, Staking and Time Vesting smart contracts. The contracts in question can be identified by the following git commit hash:

`80b7acaf8ef660cb2851ded9bc93bf41250bbc6e`

There are 10 contracts/libraries/interfaces in scope.

2. Disclaimer

The review 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 bug free status.

3. Executive Summary

There were 0 critical, 1 major, 2 minor, 53 informational/optimizational issues identified in this version of the contracts. There are **no** known compiler bugs, for the specified compiler version (0.6.11), that might affect the contracts' logic.

Considerable space is available for gas efficiency improvements. In particular, Staking implementation could borrow ideas from the StakingRewards contract by Synthetix. Solidity compiler's optimizer settings could be adjusted to account for expected number of contract executions (`runs` parameter) instead of using a default number 200. I enjoyed working with the EPNS team, especially due to their responsiveness for any inquiries during the whole engagement.

4. Critical Bugs and Vulnerabilities

No critical issues were identified during the course of review.

5. Line By Line Review

5.1. EPNS, line 112. Note, it could be useful to modify `rawAmount == uint(-1)` check in favor of `rawAmount >= uint(uint96(-1))` in order to consider anything above `uint96(-1)` to be infinite.

5.2. EPNS, line 142. Optimization, `domainSeparator` should be set in the constructor and be immutable.

5.3. EPNS, line 204. Style, line indentation is different here and in a number of other functions.

5.4. EPNS, line 204. **Major**, `mul256(balances[msg.sender], holderWeight[msg.sender])` formula is incorrect and should be replaced with the following: `mul256(balances[msg.sender]`, `sub256(block.number, holderWeight[msg.sender]))`.

5.5 EPNS, line 228. Minor, `HolderWeightChanged` event is emitted with wrong parameters. It should report information about the `holder` instead of `msg.sender`.

5.6. EPNS, line 235. Note, tokens burnt but the delegated votes remain unaffected. It is not clear if this behavior is correct or not because there are no contracts in scope that rely on the votes from EPNS.

5.7. EPNS, line 237. Optimization, `msg.sender` can never be 0x0 address, so `require(account != address(0))` assertion can be removed.

5.8. EPNS, line 264. Note, `delegateBySig()` function params design differs from those of `permit()` function. In order to have a unified style, consider passing the `owner` address instead of `nonce`.

5.9. EPNS, line 265. Optimization, `domainSeparator` should be set in the constructor and be immutable.

5.10. EPNS, line 302. Optimization, `checkpoints[account][nCheckpoints - 1]` read twice from the storage, first to fetch `fromBlock` then for `votes`. As those two values share a storage slot, it will be cheaper to read them into memory once, then fetch.

5.11. EPNS, line 357. Optimization, `balances[dst]` read from storage multiple times. Consider assigning it to a local variable once instead.

5.12. EPNS, line 363. Note, `div256(totalAmount, 2)` cannot fail so there is no point in having an error message.

5.13. EPNS, line 389. Note, this will freeze transfers on blockchains with a sub-second block time. Not a problem for Ethereum Mainnet.

5.14. Rockstar, line 25. Note, `hashCheck` could map `metadata` to a `tokenId` instead of `1` to serve the same purpose and also give the ability to find tokens by `metadata`.

5.15. CommunityVault, line 9. Optimization, `_push` can be made immutable.

5.16. YieldFarm, line 15. Optimization, `TOTAL_DISTRIBUTED_AMOUNT` can be made immutable.

5.17. YieldFarm, line 16. Optimization, `NR_OF_EPOCHS` can be made immutable.

5.18. YieldFarm, line 21. Optimization, `_token` can be made immutable.

5.19. YieldFarm, line 22. Optimization, `_communityVault` can be made immutable.

5.20. YieldFarm, line 24. Optimization, `_push` can be made immutable.

5.21. YieldFarm, line 25. Optimization, `_staking` can be made immutable.

5.22. YieldFarm, line 29. Optimization, `_genesisEpochAmount` can be made immutable.

5.23. YieldFarm, line 32. Note, it will be easier to integrate UI if `lastEpochIdHarvested` is made public. It is common to fetch data from a contract like `YieldFarm.methods.lastEpochIdHarvested(userAddress).call()` versus `YieldFarm.methods.userLastEpochIdHarvested().call({from: userAddress})`.

5.24. YieldFarm, line 33. Optimization, `epochDuration` can be made immutable.

5.25. YieldFarm, line 34. Optimization, `epochStart` can be made immutable.

5.26. YieldFarm, line 68. Style, `lastEpochIdHarvested[msg.sender] + 1` does not utilize `SafeMath.add()` which is used in other places for the same action.

5.27. YieldFarm, line 74. Minor, `MassHarvest` will always report `epochsHarvested` as 0. `lastEpochIdHarvested` needs to be saved in the beginning of the function to correctly calculate `epochsHarvested`.

5.28. YieldFarm, line 85. Note, `“Maximum number of epochs is 100”` is a misleading revert reason. Number of epochs is set in the constructor and can vary.

5.29. YieldFarm, line 130. Optimization, `lastEpochIdHarvested` can be moved to an upper layer as well as transfer in order not to update it multiple times during `massHarvest()`.

5.30. Staking, line 8. Note, `Staking` contract should inherit from `IStaking` to guarantee interface compatibility.

5.31. Staking, line 15. Optimization, `epoch1Start` can be made immutable.

5.32. Staking, line 18. Optimization, `epochDuration` can be made immutable.

5.33. Staking, line 21. Optimization, `balances[user][token]` can be removed in favor of `balanceCheckpoints[user][token][last].startBalance` as they are always equal.

5.34. Staking, line 24. Optimization, the `size` variable could utilize a smaller data type, like `uint248`, to squeeze the `set` variable in the same storage slot to make reads cheaper.

5.35. Staking, line 62. Optimization, allowance check is excessive because it will happen in the token itself.

5.36. Staking, line 64. Optimization, `balances[msg.sender][tokenAddress]` is read from storage multiple times.

5.37. Staking, line 66. Note, `token.transferFrom()` will not work for any token. It is recommended to utilize `safeTransferFrom` instead to assure support of all tokens. As long as it is used only for fully ERC20 compliant tokens, like Uniswap V2 LP, it will work correctly.

5.38. Staking, line 102. Optimization, for the `last` checkpoint, `newDeposits` are always 0. It is cheaper to read just the `startBalance`.

5.39. Staking, line 107. Optimization, `getCheckpointBalance(checkpoints[last])` executed multiple times resulting in excessive storage reads.

5.40. Staking, line 124. Optimization, `last >= 1 && checkpoints[last - 1].epochId == currentEpoch` condition is always true, at this point, and can be removed.

5.41. Staking, line 175. Note, provided commentary is partially misleading. It is true that withdrawal with no checkpoints will revert but not necessarily because of the 0 balance. Withdrawal of 0 amount will revert because `checkpoints[last]` will be out of bound.

5.42. Staking, line 186. Optimization, `checkpoints[last].newDeposits` is already 0 and does not require an update.

5.43. FundsDistributor, line 9. Optimization, `identifier` can be made immutable.

5.44. VestedReserves, line 10. Optimization, `identifier` can be made immutable.

5.45. Reserves, line 13. Optimization, `pushToken` can be made immutable.

5.46. Reserves, line 16. Optimization, `identifier` can be made immutable.

5.47. Reserves, line 37. Optimization, balance check is excessive because it will happen in the token itself.

5.48. FundsDistributorFactory, line 16. Optimization, `pushToken` can be made immutable.

5.49. FundsDistributorFactory, line 19. Optimization, `identifier` can be made immutable.

5.50. FundsDistributorFactory, line 22. Optimization, `cliff` can be made immutable.

5.51. FundsDistributorFactory, line 83. Optimization, balance check is excessive because it will happen in the token itself.

5.52. FundsDistributorFactory, line 84. Optimization, using `msg.sender` instead of `owner()` is cheaper.

5.53. TokenVesting, line 33. Optimization, `_cliff` can be made immutable.

5.54. TokenVesting, line 34. Optimization, `_start` can be made immutable.

5.55. TokenVesting, line 35. Optimization, `_duration` can be made immutable.

5.56. TokenVesting, line 37. Optimization, `_revocable` can be made immutable.

Oleksii Matiiasevych

Appendix A.

EPNS Smart Contracts Update Verification

Auditor: Oleksii Matiiasevych

1. Executive Summary

There were 0 critical, 0 major, 0 minor, 20 informational/optimizational findings were acknowledged and left unchanged in the updated version of the contracts. There are **no** known compiler bugs, for the specified compiler version (0.6.11), that might affect the contracts' logic. All the significant issues were addressed in this update, and 0 new issues were found.

2. Remaining Findings

2.1. EPNS, line 142. Optimization, `domainSeparator` should be set in the constructor and be immutable.

2.2. EPNS, line 235. Note, tokens burnt but the delegated votes remain unaffected. It is not clear if this behavior is correct or not because there are no contracts in scope that rely on the votes from EPNS.

2.3. EPNS, line 263. Note, `delegateBySig()` function params design differs from those of `permit()` function. In order to have a unified style, consider passing the `owner` address instead of `nonce`.

2.4. EPNS, line 264. Optimization, `domainSeparator` should be set in the constructor and be immutable.

2.5. EPNS, line 391. Note, this will freeze transfers on blockchains with a sub-second block time. Not a problem for Ethereum Mainnet.

2.6. Rockstar, line 25. Note, `hashCheck` could map `metadata` to a `tokenId` instead of `1` to serve the same purpose and also give the ability to find tokens by `metadata`.

2.7. YieldFarm, line 25. Optimization, `_staking` can be made immutable.

2.8. YieldFarm, line 33. Optimization, `epochDuration` can be made immutable.

2.9. YieldFarm, line 34. Optimization, `epochStart` can be made immutable.

2.10. YieldFarm, line 70. Style, `lastEpochIdHarvested[msg.sender] + 1` does not utilize `SafeMath.add()` which is used in other places for the same action.

2.11. YieldFarm, line 132. Optimization, `lastEpochIdHarvested` can be moved to an upper layer as well as transfer in order not to update it multiple times during `massHarvest()`.

2.12. Staking, line 8. Note, `Staking` contract should inherit from `IStaking` to guarantee interface compatibility.

2.13. Staking, line 21. Optimization, `balances[user][token]` can be removed in favor of `balanceCheckpoints[user][token][last].startBalance` as they are always equal.

2.14. Staking, line 24. Optimization, the size variable could utilize a smaller data type, like `uint248`, to squeeze the set variable in the same storage slot to make reads cheaper.

2.15. Staking, line 62. Optimization, `balances[msg.sender][tokenAddress]` is read from storage twice. First here, then on the line 69.

2.16. Staking, line 64. Note, `token.transferFrom()` will not work for any token. It is recommended to utilize `safeTransferFrom` instead to assure support of all tokens. As long as it is used only for fully ERC20 compliant tokens, like Uniswap V2 LP, it will work correctly.

2.17. Staking, line 101. Optimization, for the `last` checkpoint, `newDeposits` are always 0. It is cheaper to read just the `startBalance`.

2.18. Staking, line 106. Optimization, `getCheckpointBalance(checkpoints[last])` executed multiple times resulting in excessive storage reads.

2.19. Staking, line 123. Optimization, `last >= 1 && checkpoints[last - 1].epochId == currentEpoch condition` is always true, at this point, and can be removed.

2.20. Staking, line 185. Optimization, `checkpoints[last].newDeposits` is already 0 and does not require an update.