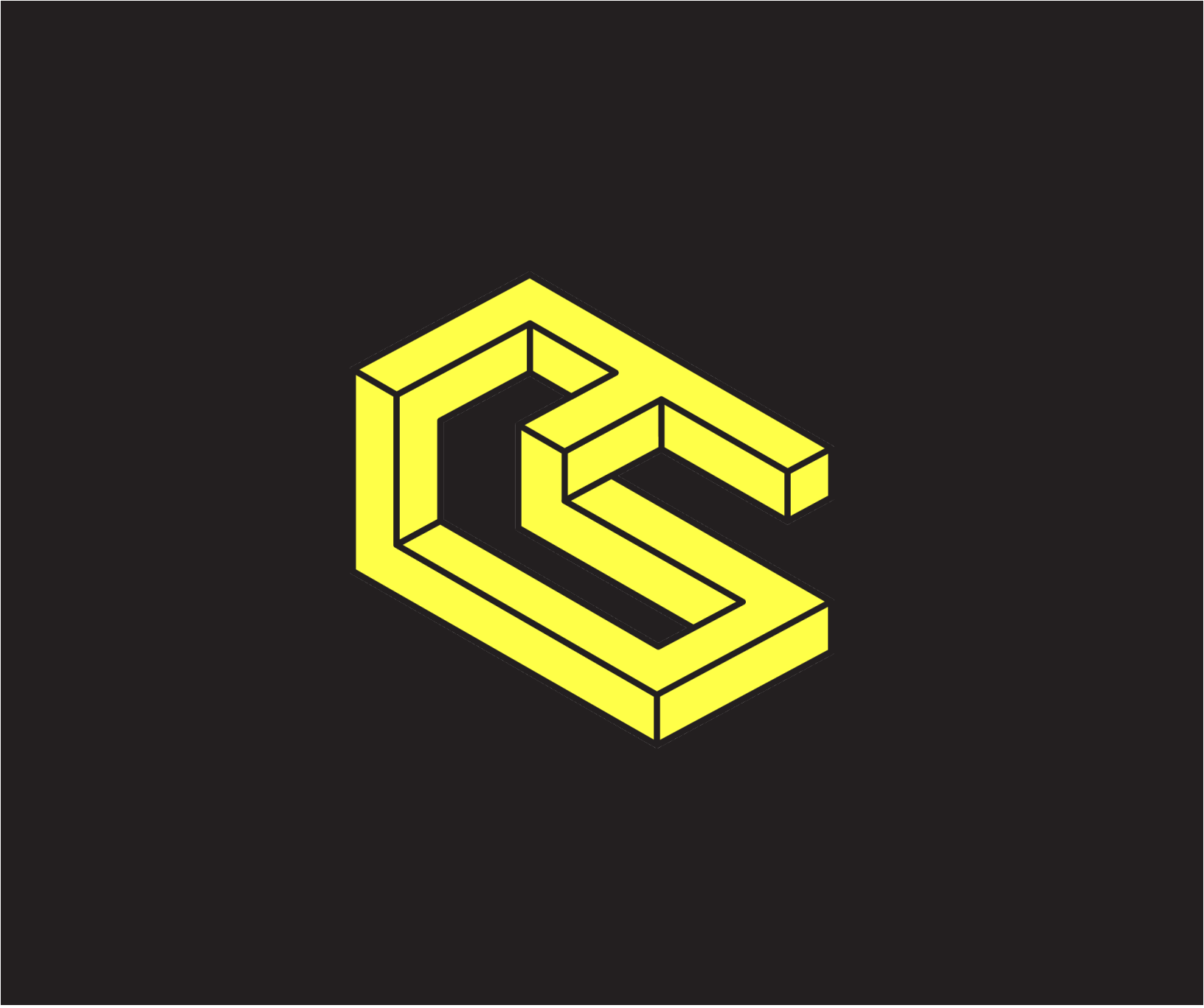

EPNS Protocol Smart Contracts Review

By: ChainSafe Systems

October 2021

EPNS Protocol Smart Contracts Review

Auditor: Oleksii Matiiasevych, Anderson Lee

WARRANTY

1. Introduction

EPNS requested ChainSafe Systems to perform a review of the EPNS Protocol smart contracts. The contracts can be identified by the following git commit hash:

`04dc260edf915aa1e81fa8e27d205bba9beb318d`

There are 9 contracts in scope.

After the initial review, EPNS team applied a number of updates which can be identified by the following git commit hash:

`95edbbf783cb862202079206b255bd8476548d4f`

Additional verification was performed after that.

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. The review 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 ChainSafe Systems.

3. Executive Summary

All the initially identified, minor and above, severity issues were fixed and are not present in the final version of the contract. No new issues were discovered in the final version.

There are **no** known compiler bugs for the specified compiler version (0.6.11), that might affect the contracts' logic.

There were 2 critical, 7 major, 7 minor, 65 informational/optimizational issues identified in the initial version of the contracts. The non-informational issues found in EPNS contracts were not present in the final version of the contracts. They are described below for historical purposes.

We are happy to continue our collaboration with the EPNS team.

4. Critical bugs and vulnerabilities

Two critical issues were identified during the review. One (5.39) could allow anyone to drain all funds from the contract, another (5.44) wouldn't allow users to claim a fair share of the interest. Interest claiming logic was removed from the final version of the contract to be introduced later with an upgrade.

5. Line-by-line review

5.1. EPNSCommProxy, line 13: Minor, `TransparentUpgradableProxy` is initialized with admin set to the same address as actors who are expected to use it (`pushChannelAdmin` and `governance`). Admin is supposed to be a different address, because admin cannot go through proxy to the implementation. Consider setting the admin to something separate right away as an additional param in the `EPNSCommProxy` constructor. You could consider using a `ProxyAdmin` contract for that, then governance could be the owner of the `ProxyAdmin`, and act through it to perform upgrades.

5.2. EPNSCoreProxy, line 20: Minor, `TransparentUpgradableProxy` is initialized with admin set to the same address as actors who are expected to use it (`pushChannelAdmin` and `governance`). Admin is supposed to be a different address, because admin cannot go through proxy to the implementation. Consider setting the admin to something separate right away as an additional param in the `EPNSCoreProxy` constructor. You could consider using a `ProxyAdmin` contract for that, then governance could be the owner of the `ProxyAdmin`, and act through it to perform upgrades.

5.3. Timelock, line 218: Note, error message '`Timelock::setDelay`' should state '`Timelock::constructor`' instead.

5.4. Timelock, line 240: Optimization, the `admin` variable is excessively read from storage, use `msg.sender` instead.

5.5. GovernorBravo, line 51: **Major**, the `admin` variable is not set at this point. This initialize function will always fail on fresh contracts.

5.6. GovernorBravo, line 77: **Major**, the '`initialProposalId != 0`' condition always fails because `initialProposalId` is always zero.

5.7. GovernorBravo, line 147: **Major**, native currency value passed in the `timelock.executeTransaction` call should already be on the `Timelock` address and not passed with execution. Or the `Timelock` should not have a payable fallback to accept native currency because it will get locked there.

5.8. GovernorBravo, line 240: Optimization, the `keccak256(bytes(name))` value should be stored as a constant.

5.9. GovernorBravo, line 343: Optimization, the `msg.sender != address(0)` condition is always true and can be removed.

5.10. EPNSCommV1, line 26: Minor, the `ReentrancyGuard` is not initialized and is not needed.

5.11. EPNSCommV1, line 117: Note, the `_notificationSender` param is not needed.

5.12. EPNSCommV1, line 123: Note, the `_notificationSender` should be replaced with the `signatory` in which case the second part of the condition, '`_notificationSender == signatory`' is not needed.

5.13. EPNSCommV1, line 138: Optimization, the `pushChannelAdmin` variable is read from storage, use the local variable `_pushChannelAdmin` instead.

5.14. EPNSCommV1, line 569: Note, the `_notificationSender` param is the `msg.sender` so it is not needed.

5.15. EPNSCommV1, line 577: Note, the `_notificationSender` param should be replaced with the `msg.sender` in which case the second part of the condition, '`msg.sender == _notificationSender`' is not needed.

5.16. EPNSCommV1, line 593: Optimization, the `_delegate` param is not needed.

5.17. EPNSCommV1, line 596: Style, missing space after the '`public`' keyword.

5.18. EPNSCommV1, line 616: Optimization, the `_delegate` param is not needed.

5.19. EPNSCommV1, line 624: Optimization, the `_delegate` param is not needed.

5.20. EPNSCommV1, line 640: Optimization, the `_delegate` param is not needed.

5.21. EPNSCommV1, line 652: Optimization, the `keccak256(bytes(name))` value should be stored as a constant.

5.22. EPNSCommV1, line 661: Optimization, the `_delegate` param is not needed.

5.23. EPNSCommV1, line 677: Optimization, the `_delegate` param is not needed.

5.24. EPNSCoreV1, line 30: Minor, the `ReentrancyGuard` and `Ownable` are not initialized and are not needed.

5.25. EPNSCoreV1, line 68: Note, the `verifiedBy` natspec has an outdated comment.

5.26. EPNSCoreV1, line 102: Note, the `mapAddressChannels` variable name seems to be unclear. Consider using `indexedChannels`, or `channelById`.

5.27. EPNSCoreV1, line 104: Optimization, the `usersInterestClaimed` variable is ever increasing and is only used for information purposes. Consider removing it from contract storage and track with an off-chain logic instead.

5.28. EPNSCoreV1, line 145: Note, the `Withdrawal` event is not used.

5.29. EPNSCoreV1, line 183: **Major**, the `governance` value should be checked instead of the `pushChannelAdmin`.

5.30. EPNSCoreV1, line 255: Optimization, the `pushChannelAdmin` variable is read from the storage excessively and should be read from the local variable `_pushChannelAdmin` instead.

5.31. EPNSCoreV1, line 323: Minor, the '`_newFees > ADD_CHANNEL_MIN_POOL_CONTRIBUTION`' condition should be greater than or equal >= because that is the initial state of the contract.

5.32. EPNSCoreV1, line 433: Note, the '`Insufficient Funds`' message seems misleading, consider changing it to '`Insufficient Deposit Amount`'.

5.33. EPNSCoreV1, line 452: Note, the `_identityList` natspec comment is missing.

5.34. EPNSCoreV1, line 458: Note, typo in the `_channelTypeLst` variable name, consider changing it to `_channelTypeList`.

5.35. EPNSCoreV1, line 469: Optimization, an unnecessary comparison `(_channelAddresses.length == _channelAddresses.length)` should be removed.

5.36. EPNSCoreV1, line 477: Optimization, could use `msg.sender` instead of the `pushChannelAdmin` variable to avoid storage reads.

5.37. EPNSCoreV1, line 607: Minor, the `channel.poolContribution` value is not updated in the `deactivateChannel` function.

5.38. EPNSCoreV1, line 608: Optimization, the `channelData` variable is only used to read one slot and write to 2 other slots. Reading it all to memory is wasteful, use a storage pointer here to save gas.

5.39. EPNSCoreV1, line 652: **Critical**, the `channel.poolContribution` value is not updated in the `reactivateChannel` function allowing to drain all funds from the contract by repeated invocations of `deactivateChannel` and `reactivateChannel`.

5.40. EPNSCoreV1, line 707: Optimization, the `channelData` variable is only used to read one slot and write to 4 slots. Reading it all to memory is wasteful, use a storage pointer here to save gas.

5.41. EPNSCoreV1, line 765: Optimization, setting the `logicComplete` value is not needed as this is the last statement in the function flow.

5.42. EPNSCoreV1, line 873: **Major**, frontrunning is possible to sandwich the swap in `swapAndTransferPUSH` function. Minimum amount out should always be specified externally.

5.43. EPNSCoreV1, line 886: Minor, there is no need to call an `approve` function for the `redeem` to succeed.

5.44. EPNSCoreV1, line 905: **Critical**, the `claimInterest` does not work as intended. Users with the equal `userHolderWeight` will get different payouts based on who was the first one to claim.

5.45. EPNSCoreV1, line 922: **Major**, there shouldn't be a division by 100 at the end of the payout calculation, as it results in the payout being 100 times smaller.

5.46. EPNSCoreV1, line 922: **Major**, for the `totalClaimableRewards` value calculation precision consider redoing it with a single division operator, ie. `totalADAIInterest*userHolderWeight/totalHolderWeight`.

5.47. EPNSCoreV1, line 987: Minor, in the case of `ChannelAction.ChannelUpdated` the `totalWeight` value calculation is wrong. Consider the case when the `count` was 3 with weights 30, 40, 50, the `totalWeight` is 120. Now the last one (50) is updated to 60. The `totalWeight` becomes 140, while it should become 130. Consider tracking the `totalWeight` in the state and calculate the `adjustedNormalizedWeight` when needed.

6. Line-by-line verification, Remaining and Acknowledged Issues

6.1. EPNSBravoProxy, line 17: Note, consider using `abi.encodeWithSelector(GovernorBravo(_logic).initialize.selector, ...)` to emphasize which logic contract is expected.

6.2. EPNSCommProxy, line 15: Note, consider using `abi.encodeWithSelector(EPNSCommV1(_logic).initialize.selector, ...)` to emphasize which logic contract is expected.

6.3. EPNSCoreProxy, line 21: Note, consider using `abi.encodeWithSelector(EPNSCoreV1(_logic).initialize.selector, ...)` to emphasize which logic contract is expected.

6.4. Timelock, line 215: Style, excessive empty line.

6.5. Timelock, line 265: Note, the `cancelTransaction` function will succeed and emit a `CancelTransaction` event even if there was no such transaction in the queue.

6.6. GovernorBravo, line 95: Optimization, consider removing the `id` from the `proposal` struct as it is always known before accessing the proposal.

6.7. GovernorBravo, line 160: Note, the proposal's vote threshold is checked for current time instead of proposal's start time. If this is an intended behavior then consider adding an explanation to the comments section stating that proposer must maintain enough votes till the end of proposal lifetime.

6.8. GovernorBravo, line 355: Optimization, the `admin` variable is read multiple times from storage in the `_acceptAdmin` function.

6.9. GovernorBravo, line 356: Optimization, the `pendingAdmin` variable is read multiple times from storage in the `_acceptAdmin` function.

6.10. EPNSCommV1, line 66: Note, the `governance` variable is not used except for assignment.

6.11. EPNSCommV1, line 235: Note, if the `_endIndex` param is lower than the `_startIndex` param then nothing will be migrated.

6.12. EPNSCommV1, line 282: Optimization, the `subscribedCount` variable is read thrice from storage.

6.13. EPNSCommV1, line 403: Optimization, the `user.subscribed[_channel]` variable i read twice from storage.

6.14. EPNSCommV1, line 408: Optimization, the `user.subscribedCount` variable is read thrice from storage.

6.15. EPNSCommV1, line 472: Optimization, the `usersCount` variable is read twice from storage.

6.16. EPNSCommV1, line 488: Style, consider using the `require` statement instead of `if` and `revert` combination.

6.17. EPNSCommV1, line 507: Note, the `else` case is not necessary as the `wallet` is `0x0` by default.

6.18. EPNSCommV1, line 573: Style, the '`(`' should be put after the function name at the same line.

6.19. EPNSCoreV1, line 64: Note, consider using an enum for the `channelState` variable.

6.20. EPNSCoreV1, line 73: Note, the `channelHistoricalZ` variable is not used.

6.21. EPNSCoreV1, line 76: Note, the `channelFairShareCount` variable is not used.

6.22. EPNSCoreV1, line 79: Note, the `channelLastUpdate` variable is not used.

6.23. EPNSCoreV1, line 97: Optimization, the `channelNotifSettings` variable is not used by contracts and is only used for information purposes. Consider removing it from contract storage and track with an off-chain logic instead.

6.24. EPNSCoreV1, line 289: Style, this line has 2 spaces indentation instead of 4.

6.25. EPNSCoreV1, line 322: Note, changing the `pushChannelAdmin` value will invalidate all verifications and will require reverification of all the primary verified channels.

6.26. EPNSCoreV1, line 367: Style, excessive indentation.

6.27. EPNSCoreV1, line 380: Style, missing indentation.

6.28. EPNSCoreV1, line 464: Style, broken formatting in this `for` loop.

6.29. EPNSCoreV1, line 468: Optimization, consider doing a single `transferFrom` and `deposit` after the loop instead of doing it with every iteration.

6.30. EPNSCoreV1, line 507: Optimization, reading the `channelsCount` variable multiple times from storage.

6.31. EPNSCoreV1, line 804-813: Optimization, if the `msg.sender` is `pushChannelAdmin`, then no other conditions need to be checked. Consider applying conditions only in case `msg.sender` does not equal `pushChannelAdmin`.

6.32. EPNSCoreV1, line 810: Optimization, the `callerVerified >= 1` condition is always true at this point.

6.33. EPNSCoreV1, line 892: Style, space is missing after the '`private`' keyword.

6.34. EPNSCoreV1, line 1014: Note, the `nxw` value could be calculated as `totalWeight.mul(x)` to not lose precision. The `nx` variable will not be needed then.

Oleksii Matiiasevych

Anderson Lee