Franklin Smart Contracts Review

By: ChainSafe Systems

June 2023

Franklin Smart Contracts Review

Auditors: Anderson Lee, Tanya Bushenyova, Oleksii Matiiasevych

WARRANTY

This Code Review is provided on an “as is” basis, without warranty of any kind, express or implied. It is not intended to provide legal advice, and any information, assessments, summaries, or recommendations are provided only for convenience (each, and collectively a “recommendation”). Recommendations are not intended to be comprehensive or applicable in all situations. ChainSafe Systems does not guarantee that the Code Review will identify all instances of security vulnerabilities or other related issues.

Introduction

Franklin Payroll Systems requested ChainSafe Systems to perform a review of the contracts used in the payroll ecosystem. The contracts can be identified by the following git commit hash:

`ab5bfe5c8db3acd7dd63619af13635d132588c4f`

There are 14 smart contracts in scope including multiple facets and libraries.
After the initial review, Franklin team applied a number of updates which can be identified by the following git commit hashes:

`556d667e81f902de04513eb12fc0108334edcb22`

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 (0.8.19), that might affect the contracts' logic.

There were 0 critical, 0 major, 2 minor, 46 informational/optimizational issues identified in the initial version of the contracts.We enjoyed working with the Franklin 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

1. FranklinPayrollFacet, line 120. Optimization, the `bulkPayWorkers()` function will excessively execute `whenNotPaused` and `onlyExecuteAdmin` modifiers on every iteration.

2. FranklinStreamingFacet, line 58: Note, the `getStreamBalance()` should probably have an `onlyExistingWorker` and `onlyApprovedTokens` modifiers.

3. FranklinStreamingFacet, line 170: Note, the `createStream()` function has an upper limit of 499 workers with stream, due to the first one being an `address(0)`.

4. FranklinStreamingFacet, line 170: Optimization, the `createStream()` function reads the `t.workersWithStream.length` value from storage multiple times.

5. FranklinStreamingFacet, line 228: Optimization, the `bulkCreateStreams()` function will execute `whenNotPaused` and `onlyStreamAdmin` modifiers on every iteration.

6. FranklinStreamingFacet, line 302: Minor, the `editStream()` function will not emit `StreamEdited` event if the current stream has already ended, and the new one starts before the next.

7. FranklinStreamingFacet, line 349: Optimization, the `endStream()` function reads the `b.current.end` value from storage multiple times.

8. FranklinStreamingFacet, line 356: Optimization, the `endStream()` function reads the `b.current.start` value from storage multiple times.

9. FranklinStreamingFacet, line 369: Optimization, the `endStream()` function reads the `b.next.end` value from storage multiple times.

10. FranklinStreamingFacet, line 373: Minor, the `editStream()` function will set the next stream start date even if the next stream does not exist.

11. FranklinStreamingFacet, line 406: Optimization, the `deleteStream()` function reads the `b.current.end` value from storage multiple times.

12. FranklinStreamingFacet, line 413: Note, the `deleteStream()` function subtracts `next.withdrawn`, which is always zero, from the `settledStreamingBalance`.

13. FranklinStreamingFacet, line 438: Optimization, the `claimPayroll()` function reads the `b.settled` value from storage multiple times.

14. FranklinStreamingFacet, line 464: Note, the `claimPayroll()` function has a code duplication for event emittance and tokens transfer, which could be moved out of the if-else block.

15. FranklinStreamingFacet, line 480: Optimization, the `_updateBalance()` function reads the `b.current.end` value from storage multiple times.

16. FranklinTreasuryFacet, line 106, 136: Note, `depositStaticFunds()` and `depositStreamingFunds()` functions have almost the same codebase. Recommend adding an internal function for the duplicated codebase.

17. FranklinUserFacet, line 110: Optimization, the `terminateWorker()` function reads the

18. FranklinUserFacet, line 112: Optimization, the `terminateWorker()` function reads the `current.end` value from storage multiple times.

19. FranklinUserFacet, line 116: Optimization, the `terminateWorker()` function reads the `current.withdrawn` value from storage multiple times.

20. FranklinUserFacet, line 123: Optimization, the `terminateWorker()` function else-if condition `current.end < _lastDay` could be replaced with just an else block.

21. FranklinUserFacet, line 136: Optimization, the `terminateWorker()` function reads the `next.start` and `next.end` values from storage multiple times.

22. FranklinUserFacet, line 140: Optimization, the `terminateWorker()` function reads the `next.withdrawn` value from storage multiple times.

23. FranklinUserFacet, line 164: Optimization, the `terminateWorker()` function will try to send 0 tokens to the worker if nothing is earned.

24. OwnershipFacet, line 6: Note, `LibDiamond.sol` is imported twice.

25. LibFranklinBalance, line 25, 74: Note, Typo. `_oken` should be replaced with `token`.

26. L ibFranklinBalance, line 50: Optimization, the `_getCurrentStreamBalance()` reads current stream entries from storage multiple times, consider storing the whole struct in memory once instead.

27. LibFranklinBalance, line 83: Optimization, the `_getNextStreamBalance()` reads next stream entries from storage multiple times, consider storing the whole struct in memory once instead.

28. LibFranklinStorage, line 4: Note, `OwnableUpgradeable` import is not used.

29. LibFranklinStorage, line 44: Optimization, some of the uint256 variables (especially `start` and `end`) could fit in a smaller integer size to be packed into one storage slot.

30. LibFranklinTreasury, line 44: Optimization, the `getStreamingTreasury()` function reads the `current.start` value from storage multiple times.

31. LibFranklinTreasury, line 46: Optimization, the `getStreamingTreasury()` function reads the `current.end` value from storage multiple times.

32. LibFranklinTreasury, line 50: Optimization, the `getStreamingTreasury()` function reads the `next.start` value from storage multiple times.

33. LibFranklinTreasury, line 52: Optimization, the `getStreamingTreasury()` function reads the `next.end` value from storage multiple times.

34. FranklinInitializer, line 6: Note, `IERC20Upgradeable` import is not used.

35. FranklinInitializer, line 7: Note, `ReentrancyGuardUpgradeable` import is not used.

36. FranklinInitializer, line 8: Note, `SafeERC20Upgradeable` import is not used.

37. FranklinInitializer, line 11: Note, `FranklinTokenWhitelist` import is not used.

38. FranklinInitializer, line 12: Note, `LibDiamond.sol` import is not used.

39. FranklinInitializer, line 31: Optimization, the `initialize()` function excessively validates the `forwarder` argument value twice.

40. FranklinScheduleForwarder, line 17: Note, Unless the contract is intended to be upgradeable, no need to inherit `Initializable`.

41. FranklinScheduleForwarder, line 36: Optimization, the `_nonceUsed` mapping could utilize a bitmap to save about 15k gas on every request.

42. FranklinScheduleForwarder, line 83: Optimization, the `execute()` function excessively validates the nonce, which is subsequently validated in the `verify()` function.

43. FranklinTokenWhitelist, line 30: Optimization, the `FranklinTokenWhitelist` contract could utilize an `AddressSet` library from OZ, which has a constant gas address removal function. Traversing through the array will quickly become expensive with array size growing.

44. FranklinTokenWhitelist, line 106: Optimization, the `removeApprovedToken()` function reads the `approvedTokens.length` value from storage multiple times.

45. ERC2771ContextUpgradeable, line 7: Note, `Initializable` import is not used.

Line by Line Review. Acknowledged Findings.

1. FranklinTreasuryFacet, line 169: Note, the `withdrawStreamingFunds()` function could reduce the streaming treasury to zero while there are active streams still present.

2. LibFranklinStorage, line 242: Note, the `adminsCanView()` modifier will not restrict external observers from accessing the data. It only enforces restrictions on other contracts calling in.

3. LibFranklinStorage, line 267: Note, the `adminOrWorkerCanView()` modifier will not restrict external observers from accessing the data. It only enforces restrictions on other contracts calling in.

Anderson Lee

Tanya Bushenyova

Oleksii Matiiasevych