NiftyKit V2 Smart Contracts Review And Verification

By: ChainSafe Systems

August 2022

NiftyKit V2 Smart Contracts Review And Verification

Auditors: Anderson Lee, Tanya Bushenyova, Oleksii Matiiasevych

WARRANTY

Introduction

NIFTYKIT, INC. requested ChainSafe Systems to perform a review of the NiftyKit V2 smart contracts. The contracts can be identified by the following git commit hash:

`a1141e7b29fb6cffc41b4c9f65159f5dfa0e59bc`

There are 10 contracts and interfaces in scope. After the initial review, NiftyKit team applied a number of updates which can be identified by the following git commit hash:

`1d71332b09dc38e5e14372bcca77f34cd03f44c4`

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

All the initially identified minor, and above, severity issues were fixed and are not present in the final version of the contract. There are no known compiler bugs for the specified compiler version (0.8.15), that might affect the contracts' logic. There were 0 critical, 0 major, 1 minor, 19 informational/optimizational issues identified in the initial version of the contracts. All the findings, except of two optimization ones, were fixed and are not present in the final version. They are described below for historical purposes. An additional issue, that could block minting new tokens in a collection, was identified and fixed by the NiftyKit team during the engagement. We are looking forward to future engagements with the NiftyKit team.

Critical Bugs and Vulnerabilities

No critical issues were identified.

Line by Line Review. Fixed Issues

1. DropCollection, Note: Consider adding a constructor with `initialize()` in it to avoid initialization of the implementation contract by anyone.

2. DropCollection, line 6: Note, the `ERC2981Upgradeable` import is not used.

3. DropCollection, line 17: Note, the `MerkleProofUpgradeable` using-declaration is not used.

4. DropCollection, line 34: Note, the `onlyMintable()` modifier has an incorrect revert message "Greater than 0", it should be something like "Must be greater than 0" or "Quantity is 0".

4. DropCollection, line 34: Note, the `onlyMintable()` modifier has an incorrect revert message "Greater than 0", it should be something like "Must be greater than 0" or "Quantity is 0".

5. DropCollection, line 74: Optimization, in the `presaleMint()` function, the `_mintCount[_msgSender()]` variable is read multiple times (in `presaleMint()` function and in `onlyMintable` modifier, also in `_purchaseMint()` function).

6. DropCollection, line 93: Note, in the `batchAirdrop()` function arrays length validation, the error message is missing.

7. DropCollection, line 96: Note, no checks are performed for mint amount in the `batchAirdrop()` function. Incrementing `_mintCount[to]` could make sense for future checks during the sale.

8. DropCollection, line 158: Optimization, in the `_purchaseMint()` function, the `quantity > 0` validation is performed twice, in the `onlyMintable()` modifier and in the `_purchaseMint()` function.

9. DropCollection, line 158: Note, the `_purchaseMint()` function quantity validation revert message is incorrect, it should be "Must be greater than 0".

10. TokenCollection, Note: Consider adding a constructor with `initialize()` in it to avoid initialization of the implementation contract by anyone.

11. TokenCollection, line 9: Note, the `ERC2981Upgradeable` import is not used.

12. TokenCollection, line 76: Optimization, in the `redeem()` function the `redeemableAt(redeemableId)` is called twice(every time reading from storage), second time inside the `_redeem()` function.

13. BaseCollection, line 40: Optimization, in the `withdraw()` function the `_niftyKit` variable is read from storage three times.

14. BaseCollection, line 41: Optimization, in the `withdraw()` function the two consecutive calls to `_niftyKit` contract could be combined into one function to save gas.

15. NiftyKitV2, Note: Consider adding a constructor with `initialize()` in it to avoid initialization of the implementation contract by anyone.

16. NiftyKitV2, line 163: Minor, in the `_createCollection()` function, if `msg.sender` has too many tokens (too many entries in `dropKitPass` contract) then `_dropKitPass.getFeeRateOf()` will fail (because gas needed for the "for" loop can exceed the block gas limit). Collection creation will fail for this user.

17. Redeemables, line 31: Note, the `MerkleProofUpgradeable` using-declaration is not used.

18. Redeemables, line 121: Optimization, in the second `_redeem()` function the `redeemableAt(redeemableId)` is called twice (every time reading from storage).

Line By Line Verification. Acknowledged Findings

1. TokenCollection, line 92: Optimization, in the `redeem()` function the `redeemableAt(redeemableId)` is called twice (every time reading from storage), second time inside the `_redeem()` function.

2. Redeemables, line 91: Optimization, in the `_redeem()` function, the `_redeemedByWallet[redeemableId][_msgSender()]` variable is read from storage twice.

[illegible]

Oleksii Matiiasevych

[illegible]

Tanya Bushenyova

[illegible]

Anderson Lee