LookFor

Smart Contracts Audit

Viacheslav Vozniuk, Paul Adamenko,
Yaroslav Tumanov

October, 2023

Production-ready contracts

6/10

Email: info@4irelabs.com

Website: 4irelabs.com

CONFIDENTIAL

This document and the Code Review it references is strictly private, confidential and personal to its recipients and should not be disclosed, copied, distributed, or reproduced in whole or in part, not passed to any third party.

1. **Introduction.** LookFor requested 4irelabs a review of the contracts implementing their ERC721 token and staking.
   There are 3 contracts source code in scope:
   - LFG.sol
   - NFTStaking.sol
   - LookforMultiSig.sol

2. **Warranty.** Audit is provided on an "as is" basis, without warranty of any kind, express or implied. The Auditor does not guarantee that the Code Review will identify all instances of security vulnerabilities or other related issues.

3. **Executive Summary.** Provided contracts realize ERC721 mintable tokens and their staking mechanism. Only whitelisted users could mint tokens. The contracts are upgradeable, so the code is not immutable and could be updated at any time. The contracts implement a custom multi-sig wallet. It could be more secure to use well-known multi-sig solutions to reduce risks. There are no known compiler bugs, for the specified compiler versions, that might affect the logic of the contracts.

4. **Critical Bugs and Vulnerabilities.** 3 Major issues have been found (LFG.sol 1, 2. NFTStaking.sol 2).

5. LINE BY LINE REVIEW

LFG.sol

1. **INFO** LFG.sol

Contract is upgradable and because of this centralized risks appear. Admin could upgrade any vesting or token contract and use it for selfish purposes.

2. **MAJOR** LFG.sol Line 196

During public mint, when checking the **singleAmount**, only the changed NFTs are taken into account, and the quantity of NFTs being minted is not considered. In other words, with an account limit of, for example, 10 NFTs, you can mint a maximum of 19 NFTs initially 9 and then 10.

Recommendation: Replace

mintedAmount >= publicMintInfo.singleAmount ||

with

mintedAmount + num >= publicMintInfo.singleAmount ||

3. **MAJOR** LFG.sol Line 232

During whitelist mint, when checking the **singleAmount**, only the changed NFTs are taken into account, and the quantity of NFTs being minted is not considered. In other words, with an account limit of, for example, 10 NFTs, you can mint a maximum of 19 NFTs initially 9 and then 10.

Recommendation: Replace

mintedAmount >= freeMintInfo.singleAmount ||

with

mintedAmount + num >= freeMintInfo.singleAmount ||

4. **LOW/INFO** LFG.sol **Line 89** **function _getRandomNum**

When calculating a random number, blockchain parameters are used, so these values can be tampered with.

Recommendation: If the logic of contracts or business operations significantly depends on these values, it is advisable to use random numbers from oracles; otherwise, you can leave it as is.

5. **LOW** LFG.sol **Line 43** **function _useRandomAvailableTokenId**

When working with an array of NFT indexes, the *Fisher-Yates shuffle* algorithm is used. Consequently, as a result, the distribution of NFTs will be skewed towards the end of the array since when duplicates occur, indexes are taken from the end of the array.

Recommendation: Use a different algorithm for working with the array of NFT indexes.

6. **LOW** LFG.sol **Line 122** **function setupFreeMintInfo**

During initialization and updating of **freeMintInfo**, it's incorrect to specify **availableAmount** since this value depends on the number of NFT mints and **maxAmount**.

Recommendation: Remove the **availableAmount** parameter from the function's parameters and use **maxAmount** for its initialization.

7. **LOW** LFG.sol **Line 144** **function setupPublicMintInfo**

During initialization and updating of **publicMintInfo**, it's incorrect to specify **availableAmount** since this value depends on the number of NFT mints and **maxAmount.**

Recommendation: Remove the **availableAmount** parameter from the function's parameters and use **maxAmount** for its initialization.

8. **LOW** LFG.sol Line 122 **function setupFreeMintInfo**

The function lacks a check for the **newFreeMintInfo.singleAmount** parameter, and if it's set to zero, the whitelist mint functionality will be blocked.

Recommendation: Add a check for the parameter to ensure it's greater than zero.

9. **LOW** LFG.sol Line 144 **function setupPublicMintInfo**

The function lacks a check for the **publicMintInfo.singleAmount** parameter, and if it's set to zero, the public mint functionality will be blocked.

Recommendation: Add a check for the parameter to ensure it's greater than zero.

10. **INFO** LFG.sol Line 109 **function setupLockInfo**

When setting the contract parameters, there is no event generation.

Recommendation: Add event generation.

11. **INFO** LFG.sol Line 122 **function setupFreeMintInfo**

When setting the contract parameters, there is no event generation.

Recommendation: Add event generation.

12. **INFO** LFG.sol Line 144 **function setupPublicMintInfo**

When setting the contract parameters, there is no event generation.

Recommendation: Add event generation.

13. **INFO** LFG.sol **NatSpec documentation missing**

If you started to comment on your code, also comment on all other functions, variables, etc

14. **GAS OPTIMIZATION** LFGStorage.sol **Line 11**

The array for storing NFT indices, **_availableRemainingTokens**, is initialized with a size of **65536**, even though the number of NFTs we have is **MAX_SUPPLY = 3000**, and only a portion of the array **[0 .. MAX_SUPPLY - 1]** is used.

Recommendation: Initialize the array with the required size **uint16[MAX_SUPPLY] internal _availableRemainingTokens;.**

15. **GAS OPTIMIZATION** LFGStorage.sol **Line 16** **struct MintInfo**

For gas economy, the struct **MintInfo** can be written as

```
struct MintInfo {
    uint32 startTime;
    uint32 endTime;
    uint32 singleAmount;
    uint32 maxAmount;
    uint32 availableAmount;
}
```

NFTStaking.sol

1. **INFO** NFTStaking.sol

Contract is upgradable and because of this centralized risks appear. Admin could upgrade any vesting or token contract and use it for selfish purposes.

2. **MAJOR** NFTStaking.sol **Line 175** **function depositNft**

Inconsistency with the Docs. During NFT deposit, ownership verification is only performed when **position.stakedAmount == 0**, meaning only during the first deposit. Subsequently, any account can add a deposit without having the NFT.

Recommendation: Remove the condition **position.stakedAmount == 0** so that ownership verification occurs with each deposit.

3. **LOW** NFTStaking.sol Line 23 **function initialize**

There is no validation of the **token_** and **nftContractAddress_** parameters during the contract initialization.

Recommendation: Add validation for valid addresses.

4. **LOW** NFTStaking.sol Line 36 **function initialize**

The variable **lastRewardedBlock** doesn't serve any meaningful purpose.

```
uint256 lastRewardedBlock = startBlockNumber_;
nftPool = Pool({
nftContract: ERC721EnumerableUpgradeable(nftContractAddress_),
stakedAmount: 0,
lastRewardedBlock: lastRewardedBlock,
accumulatedRewardsPerShare: 0
});
```

Recommendation: Remove the **lastRewardedBlock** variable.

5. **LOW** NFTStaking.sol Line 203 **function claimNft**

In the function for receiving rewards from the NFT deposit, the logic for withdrawing locked funds from the **LFG.sol** contract has been added. This makes the function's behavior illogical and can be misleading.

Recommendation: Extract the logic for withdrawing locked funds from the **LFG.sol** contract into a separate function.

6. **INFO** NFTStaking.sol Line 101 **function setupNftLevel**

When setting the level for an NFT, there is no event generation.

Recommendation: Add event generation.

7. **INFO** NFTStaking.sol Line 332 **function pendingRewards**

When obtaining an NFT **Position**, it is possible to perform a check on it and return 0 immediately if the **Position** is zero.

3. **LOW** NFTStaking.sol Line 23 **function initialize**

There is no validation of the **token_** and **nftContractAddress_** parameters during the contract initialization.

Recommendation: Add validation for valid addresses.

4. **LOW** NFTStaking.sol Line 36 **function initialize**

The variable **lastRewardedBlock** doesn't serve any meaningful purpose.

```
uint256 lastRewardedBlock = startBlockNumber_;
nftPool = Pool({
nftContract: ERC721EnumerableUpgradeable(nftContractAddress_),
stakedAmount: 0,
lastRewardedBlock: lastRewardedBlock,
accumulatedRewardsPerShare: 0
});
```

Recommendation: Remove the **lastRewardedBlock** variable.

5. **LOW** NFTStaking.sol Line 203 **function claimNft**

In the function for receiving rewards from the NFT deposit, the logic for withdrawing locked funds from the **LFG.sol** contract has been added. This makes the function's behavior illogical and can be misleading.

Recommendation: Extract the logic for withdrawing locked funds from the **LFG.sol** contract into a separate function.

6. **INFO** NFTStaking.sol Line 101 **function setupNftLevel**

When setting the level for an NFT, there is no event generation.

Recommendation: Add event generation.

7. **INFO** NFTStaking.sol Line 332 **function pendingRewards**

When obtaining an NFT **Position**, it is possible to perform a check on it and return 0 immediately if the **Position** is zero.

8. **INFO** NFTStaking.sol **Line 12**

There is a commented import statement for **console.log:**

// **import "forge-std/console.sol";**

Recommendation: Remove all of the commented code.

9. **INFO** NFTStaking.sol **NatSpec documentation missing**

If you started to comment on your code, also comment on all other functions, variables, etc.

10. **GAS OPTIMIZATION** NFTStaking.sol **Line 219, Line 260**

Multiple calls of **safeTransfer throw _claim**. It will be cheaper if the transfer is in one single **safeTransfer** call that will be the sum of all transfer amounts that exist now

Recommendation: Call **safeTransfer** method once and after the cycle

LookforMultiSig.sol

1. **INFO** LookforMultiSig.sol **Line 189**

Events should be emitted here. It is an important change for multisig smart contract, so event about this could be added.

2. **INFO** LookforMultiSig.sol **Line 193**

Need to set up current require before the cycle. And modify it, to check that the incoming owner addresses array is acceptable for function execution.

3. **INFO** LookforMultiSig.sol **Line 201**

Better add a few validations. First - for the address is not zero and second - for checking the amount of tokens, before transfer.