Faraland
Security Analysis

Produced for

By

MOONPOOL

Overview

Abstract

In this report, we consider the security of the Faraland project. Our main task is to find and describe security issues in the smart contracts of the platform to the team.

Limitations of the report

Broadly speaking, the assessments can not uncover all vulnerabilities of the given smart contracts, thus, it is not guaranteed that the system is secured even if no vulnerabilities are found. The focus of the assessments was limited to only **4** given smart contracts and corresponding interfaces, other contracts were excluded (including external libraries or third party codes).

Summary

The initial audit showed **1 high** severity issue, **1 medium** severity issue and **10 low** severity issues.

After the first security audit report, the team has fixed all issues.

Recommendations

We recommend the team to fix all issues, especially for high and medium ones, as well as test coverage to ensure the security of the contracts.

We strongly recommend the team to add meaningful and detailed comments to the source codes. It will help to understand the logic better and easier to audit the correctness of the codes.

Assessment Overview

Scope of the audit

Source codes for the audit were taken from the commit hash ***bd986ac1c6f4ae0ba9bacd28242724604974bedb*** in their private repository on April 25th, 2021. The assessment was performed on only 4 files in the foldes:

1. **Equipment.sol**
2. **EquipmentVendor.sol**
3. **MoonKnight.sol**
4. **StakingExpPool.sol**

and corresponding interfaces: IEquipment.sol, IEquipmentVendor.sol, IMoonKnight.sol and IStakingExpPool.sol.

Other source codes are out of scope for this report.

After the first iteration, the team has updated the source code to the commit hash ***5820a05a2ad34e6f49f9410700ce1c95024cf2c6*** in their private repository on May 5th, 2021.

System Overview

The system overview is based on the provided documentation here.

MoonKnight Contract (ERC721)

1. **claimMoonKnight**: Claim a Knight during NFT sale. All Knights will be randomly distributed once the sale ends by startingIndex with pre-generated images proven by **MOON_KNIGHT_PROVENANCE**
    - Each wallet can only buy **10** NFTs at most.
    - Stop when all **20,000** units have been distributed.
    - **90%** funds goes to the owner, **10%** remaining goes to the floor price of NFT.
2. **changeKnightName**:
    - Must be a unique name.
    - Must spend a sufficient amount of FARA token defined in serviceFeeInToken. The FARA goes to the owner.
    - Names should be alphanumeric and spaces without leading or trailing space.
3. **attachSymbolToKnight**: Allow users to attach a symbol to their Knight (Ex. BTC, ETH)
    - Only maximum **5** Knights per symbol.
    - Must spend a sufficient amount of FARA token defined in serviceFeeInToken. The FARA goes to the owner.
    - Cannot be re-attached
    - Must be valid symbol (cannot be more than **5** characters, cannot contain spaces)
4. **equipItems**: equip items (ERC1155) to a Knight, burn the items after equipped and attach them to corresponding Knight.
    - itemId cannot be 0.
    - itemType cannot be **SKILL_BOOK**.
    - itemIds cannot contain duplicate item type.
5. **removeItems**: remove items from a Knight, mint the items back to users.
    - itemId cannot be 0.
    - itemType cannot be **SKILL_BOOK**.
    - itemIds cannot contain duplicate item type.
6. **addFloorPriceToKnight**: allow the user to manually add BNB into the floor price of a Knight.

    - Must spend a sufficient amount of FARA token that is increased based on their value added.
    - Cannot manually add floor price by more than the number defined in floorPriceCap. (other methods to increase floor price like trading or duel can still be accumulated)
7. **sacrificeKnight**: burn a Knight to exchange for its floor price in ETH.
8. **list**: the owner can call this function to put an order to sell a Knight.
    - a Knight cannot be listed with a price under its floor price.
9. **delist**: the owner can cancel their listing knight.
10. **buy**: a buyer can call this function to buy a listed Knight.
    - a percentage of total traded value goes to owner as defined in marketFeeInBps
    - a percentage of total traded value goes to the knight's floor price itself as defined in floorPriceInBps
11. **offer**: User can send an amount of BNB as escrow to offer for a Knight.
12. **takeOffer**: the owner can call this function to take an offer.
    - a percentage of total traded value goes to owner as defined in marketFeeInBps
    - a percentage of total traded value goes to the knight's floor price itself as defined in floorPriceInBps
13. **cancelOffer**: cancel the order and get back their BNB.
14. **learnSkill**: a Knight can learn a skill (ERC1155).
    - Burn the skill book permanently.
15. **adoptPet**: attach a Pet to a Knight. (will be implemented later, pet will be ERC721 with generic mechanism like Axies)
16. **abandonPet**: de-attach a Pet from a Knight. (will be implemented later)
17. **generateKnight**: operator contract can mint Knight.
18. **levelUp**: operator contract can level up a knight.
19. **finalizeDuelResult**: the floor price of the winning Knight and losing Knight will be calculated based on the duel result.

Equipment Contract (ERC1155)

1. **createItem**: create a new item by name, maxSupply, itemType and rarity.
2. **addNextTierItem**: allow an item to advance to the next tier by burning enough items.
3. **upgradeItem**: burn a sufficient amount of items to upgrade to the next tier.
    - Items must be upgradeable.
    - Must have enough amount of items defined in upgradeAmount
    - Must spend sufficient amount of FARA token defined in upgradeFeeInToken
4. **rollEquipmentGacha**: get random items based on the rarities.
    - **40%** for COMMON
    - **30%** for UNCOMMON

    - **20%** for RARE
    - **8.5%** for EPIC
    - **1%** for LEGENDARY
    - **0.5%** for MYTHICAL.
5. **mint**: using an EquipmentVendor contract.
6. **putItemsIntoStorage**: using in MoonKnight contract.
7. **returnItems**: using in MoonKnight contract.
8. **getItem**: get item information.
9. **getItemType**: get a type of given item.
10. **isOutOfStock**: check if an item is sold out.

EquipmentVendor Contract

(Should remain unverified in etherscan since there is no real random involved as of now)

1. **addItemsToVendor**: add a list of item ids to a gacha machine categorized by vendorId.
    - All items must be 1st tier items (not an upgraded version).
2. **removeItemsFromVendor**: remove a list of item ids from a gacha machine.
3. **mintRandomItems**: serving rollEquipmentGacha.
    - Only Equipment contracts can call this function.
4. **getVendorItemIds**: get all item ids in a given vendor.

StakingExpPool Contract

Staking FARA token to earn EXP for a given knight. Anybody can add their FARA to stake for a knight (not only knight owner)

1. **stake**: Stake FARA to a Knight for farming EXP. Also harvest any EXP earned.
2. **unstake**: Unstake FARA from a Knight. Also harvest any EXP earned.
3. **convertExpToLevels**: Harvest EXP earned and convert them to levels. Maximum 10 levels can be increased at a time.
4. **exit**: Withdraws all fund from every Knights under staking.
5. **emergencyWithdraw**: Withdraws all fund from a Knight in case of emergency.
6. **getExpEarned**: get EXP earned so far of a Knight attached to an address.
7. **balanceOf**: get staked balance of FARA in a Knight attached to an address.

Findings

The initial report showed **1 high** severity issue, **1 medium** severity issue and **10 low** severity issues.

After the first security audit report, the team has fixed all issues.

MoonKnight.sol

[Fixed] [High] **addFloorPriceToKnight**: Floor price can be over price cap

Contract only checks if the current floor price is lower than cap, floor price after function call could still be higher.

Comment: This issue has been fixed and not presented in the code.

[Fixed] [Low] The owner can take his/her listing items

By design, the owner can not make an offer for his/her own items, however, still be able to take his/her listing items.

Comment: This issue has been fixed and not presented in the code.

[Fixed] [Low] Use SafeERC20 for ERC20 token

Instead of checking for **transferFrom** result, consider using SafeERC20 for **acceptedToken**.
Lines 152, 169, 203.

Comment: This issue has been fixed and not presented in the code.

[Fixed] [Low] **learnSkill:** Should check if sender is owner of the knight

Should check if the sender is the owner of a knight to avoid a player unintentionally losing a skill book for the knight he does not own.

Comment: This issue has been fixed and not presented in the code.

[Fixed] [Low] **modifier**: Create modifier to check for fee in token

Since the fee in token is checked and collected in multiple functions, consider creating a modifier to check before executing the function, and collecting tokens at the end.

Comment: This issue has been fixed and not presented in the code.

[Acknowledged] [Low] Expensive cost for on-chain orderbook

The contract is using an on-chain orderbook mechanism, it is really expensive, especially when updating orders frequently. Can explore the off-chain sign transaction mechanism.

Comment: The team is aware of this issue

StakingExpPool.sol

[Fixed] [Low] **getExpEarned**: Should using precision number to calculate

Currently not using precision to calculate which make it might take very long time to earn single EXP

Comment: This issue has been fixed and not presented in the code.

[Fixed] [Low] Consider adding an emergency withdrawal function.

To be safe for users, please add an emergency withdrawal function with the simplest logic.

Comment: This issue has been fixed and not presented in the code.

EquipmentVendor.sol

Note: This contract is supposed to be kept as a private contract.

[Fixed] [Medium] **mintRandomItems:** revert if one of the items has been out of stock

equipmentContract.mint(account, randItemId, 1) will be reverted if the item has been out of stock. Should find a solution to prevent it from reverting, unless it is intentional.
Consider to check if the item is not out of stock before calling mint function.

Comment: This issue has been fixed and not presented in the code.

[Fixed] [Low] Uncheck for returned values from add/remove items when using EnumerableSet.

In the EnumerableSet operations, **add** and **remove** return a boolean, but the returned values are not verified in the code.

Comment: This issue has been fixed and not presented in the code.

Equipment.sol

[Fixed] [Low] Use SafeERC20 for ERC20 token

Instead of checking for **transferFrom** result, consider using SafeERC20 for **acceptedToken**.
Lines 110, 123.

Comment: This issue has been fixed and not presented in the code.

[Partial-Fixed] [Low] [Documentation] Clear comment for **rollEquipmentGacha** function that doesn't allow the sender as a contract

Should add a clear comment that the function only accepts calls from normal addresses.

Comment: The team has added more revert messages to show the logic.

Testing

The team has provided some tests but we recommend adding full tests coverage to ensure the logic works as expected and the security of the smart contracts.