Wanaka Security Analysis

Overview Abstract

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

Limitations and use 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 given smart contracts, other contracts were excluded (including external libraries or third party codes).

Summary

We have found 3 high severity issues, 3 medium severity issues and 1 low severity issue.

We have discussed closely with the team about the design & implementation of the source code. The team has taken our suggestions seriously and made changes accordingly.

All high and medium issues have been adjusted and not presented in the latest source codes.
Most of the low issues have been adjusted.

Recommendations

We recommend the team to fix all issues, as well as test coverage to ensure the security of the contracts.

Assessment Overview

Scope of the audit

Source codes for the audit were initially taken from the commit hash 5340da2474099525d04e50a5a2b27009717b5471. In-scope contracts for audit:

- All contracts in the ***contracts/*** folder, including: *farm*, *wana-staking*, *BaseFarm.sol*

Other source codes are out of scope for this report.

After the initial report for the source codes above, the team has fixed all findings. The final source codes for the audit are taken from the commit hash 9f23d7d6b9f3710bed1d10b9641a912bfab15878

System Overview

Wanaka Farm setting is inspired by a dreamy beautiful town from the South Island of New Zealand, called Wanaka. The town of Wanaka is situated at the southern end of Lake Wanaka, surrounded by gorgeous natural beauty of mountains, lakes, and forest. The game feel will incorporate as much sight and sound of Wanaka to evoke natural beauty and immersive relaxing experience as possible.

At Wanaka Farm, players will immerse themselves in the role of a happy farmer by cultivating lands, crop farming, breeding pets, and decorating their own virtual
land. From then, each participant will contribute to building a whole metaverse combining multiple unique customized farms.

Wanaka provides lands and several in-game assets including Seed, Growing, Harvested, Product and Building items which are tradeable on Wanatrade - a Wanaka's marketplace.

The full details of the system and implementation can be found in the Green Paper.

Findings

We have found **3** high severity issues, **3** medium severity issues and **1** low severity issues.

We have discussed closely with the team about the design & implementation of the source code. The team has taken our suggestions seriously and made changes accordingly.

All high and medium issues have been adjusted and not presented in the latest source codes.
Most of the low issues have been adjusted.

[Fixed] [High] BaseFarm: overflow or run out of gas with high number of rounds

In the **BaseFarm**, it is re-calculating the multiplier from the first round until the latest. The for loop can either run out of gas when the number of rounds is too high, or get overflow for **rateConfig.reducingRate** ** **i** or **100** ** **i** where **i** is around 39.

Suggestion: Can define a base rate value **baseRate**, and for each iteration, the **baseRate** is re-calculated by:

baseRate = baseRate * rateConfig.reducingRate / 100;

This will prevent the overflow issues, however, the team has stated that the number of rounds will be small (1 round is 1 month period).

Comment: This issue has been adjusted with the suggestion.

[Fixed] [High] Vesting: Wrong _totalAmount for user

In the function **addVesting(user, amount)** from the **Vesting** contract, it is calculated **_totalAmount[msg.sender] += amount**, where it should be **_totalAmount[user] += amount** since the function is updating data for the **user**.

Comment: This issue has been adjusted.

[Fixed] [High] Vesting: Wrong check for a vestingInfo is inactive

In the **_claimVesting** function from the **Vesting** contract, it is checked that if **_userToVestingList[_user][_index].amount == claimableAmount**, then the vestingInfo is inactive (already claimed all amount).

It is wrong as it should check if the total reward amount is equal to the total claimed amount, instead of the claimable amount.

Suggestion: check if **_userToVestingList[_user][_index].amount == _userToVestingList[_user][_index].claimedAmount**.

Comment: This issue has been adjusted.

[Fixed] [Medium] Vesting + WanaFarm: Run out of gas if a user has too many vestings/lockRewards.

In the **Vesting** contract, functions like **claimTotalVesting, getVestingTotalClaimableAmount** are iterating through all a user's vestings, which can run out of gas when the user has too many vestings, especially for claim vestings function, where it costs lots of gas to update states and transfer rewards to the user.

It is similar to the contract **WanaFarm** when the user's lock reward list has too many elements, the **claimTotalReward** and **getTotalClaimableAmount** will run out of gas.

Suggestions: add start & end index to both functions to execute only from **startIndex** to **endIndex** of the user's vestings.

Comment: This issue has been adjusted.

[Fixed] [Medium] Wrong index for checking reducing rate

In the **WanaFarm & Farm** contracts, it is checking reducing rate with:

require(_rateParameters[2] > 0 && _rateParameters[1] < 100, ...)

This is wrong as the second check should be **_rateParameters[2] < 100**.

Comment: This issue has been adjusted.

[Fixed] [Medium] Should check for the vestings/lockRewards list to be empty to avoid underflow issue

In the **WanaFarm & Vesting** contracts, it is not checked if the **_userToVestingList** or **userToLockRewardList** to be empty in the claim or get a claimable amount.

Suggestion: Should check for the list to be empty before executing the logic to prevent underflow.

Comment: This issue has been adjusted.

[Partial - Fixed] [Low] Remove redundant imports, casting, unused attributes etc.

1. **Farm**
    a. **FarmFactory** is redundant.

2. **FarmGenerator**
    a. Should change **_rateParameters** from uint256[] to uint256[4], also remove the check for **_rateParameters.length** to be equal to 4.
    b. Should change **_vestingParameters** from uint256[] to uint256[2], also remove the check for **_vestingParamters.length** to be equal to 2.
3. **Vestings**
    a. Redundant **Ownable** import.

b. **token** and **farmAddress** should be immutable since they are assigned only once in the constructor.

4. **WanaFarm**
    a. Redundant castings: redundant casting from msg.sender to address(msg.sender).
    b. Should rename **tokenAddress** to **stakeToken**, so it is easier to understand the logic.

Suggestion: Remove redundant imports, functions, variables, castings, etc.

Comment: Most of the issues have been adjusted, some are kept for information purposes.

Testings

The team has provided tests for all contracts, they do not provide 100% full coverage yet. We strongly recommend the team to add full test coverage to ensure the security of the codes.