#HashEx
BLOCKCHAIN SECURITY

Manufactory LandStaking

smart contracts final audit report

June 2022


hashex.org


contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the Manufactory team to perform an audit of their smart contract. The audit was conducted between 14.06.2022 and 15.06.2022.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code is available at 0x37F2CBe7859cC458590b109cDC27fAb63404DD8e.

2.1 Summary

Project name	Manufactory LandStaking
URL	https://manufactory.gg
Platform	Aurora
Language	Solidity

2.2 Contracts

Name	Address
LandStaking	

3. Found issues

Medium	1 (20%)
Low	3 (60%)
Info	1 (20%)

C1. LandStaking

ID	Severity	Title	Status
C1-01	Medium	Withdrawing and changing rewards by owner	Open
C1-02	Low	Gas optimizations	Open
C1-03	Low	Lack of validation of input parameters	Open
C1-04	Low	Rewards sources	Open
C1-05	Info	Typos	Open

4. Contracts

C1. LandStaking

Overview

The contract allows the staking of MNFT-Lands ERC1155 tokens for 30, 60, or 90 days. The reward rate of MNFT ERC20 tokens is fixed, and rewards are distributed among users according to staked token IDs and locking periods.

Issues

C1-01 Withdrawing and changing rewards by owner

Medium Open

a. The contract owner has the ability to withdraw all reward tokens from the contract using the `withdrawRewardsToken()` function at any time. Thus, users may not receive the expected rewards, since they will not be in the contract.

```
function withdrawRewardsToken(uint256 amount, address token)
    external
    onlyOwner
{
    IERC20(token).transfer(msg.sender, amount);
    emit RewardTokensWithdrawn(msg.sender, amount);
}
```

b. The contract owner has the ability to change the reward token using the `setRewardsToken()` function. This can lead to the fact that the user, expecting to receive one token, will receive a completely different one (with a different value).

```
function setRewardsToken(IERC20 _rewardsToken) external onlyOwner {
    rewardsToken = _rewardsToken;
    emit RewardsTokenSet(msg.sender, address(_rewardsToken));
}
```

Recommendation

a. Consider adding a cooldown period for withdrawing rewards by the owner.

b. To prevent user frustration about changed reward token, the `getReward()` function may receive the expected reward token address in parameters and ensure its equality to the stored one (preventing possible front-run from the owner).

We also recommend to transfer ownership to a Timelock-like contract, which automatically mitigates the first case of the cooldown period.

C1-02 Gas optimizations

Low Open

Multiple reads from the storage in the `updateReward()` modifier and the `stake()` function.

C1-03 Lack of validation of input parameters

Low Open

a. The contract constructor does not check the address `stakeToken` for a non-zero value.

b. The function `setRewardsToken()` does not check the address `_rewardsToken` for a non-zero value or to any type of IERC20 compliance.

C1-04 Rewards sources

Low Open

Users should receive rewards for their staking. But there is no explicit guarantee that the contract has enough rewards for users at any time. A possible solution is to track the end-of-reward timestamp as the reward rate is immutable. In that case, the `earned()` function should be modified too.

C1-05 Typos

Info Open

Typos in comments in L126 and L193.

5. Conclusion

1 medium and 2 low severity issues were found.

The reviewed contract is highly dependent on the owner’s account. Users using the project have to trust the owner and that the owner's account is properly secured.

This audit includes recommendations on improving the code and preventing potential attacks.

Appendix A. Issues' severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

 contact@hashex.org

 @hashex_manager

 blog.hashex.org

 linkedin

 github

 twitter

#HashEx
BLOCKCHAIN SECURITY