CYBER UNIT Technologies

Smart Contract Audit

Final Report

Date: August 31, 2021
Report for: Hord / DcentraLab
By: Cyber Unit Technologies

This document may contain confidential information about IT systems and intellectual property of the customer as well as information about potential vulnerabilities and methods of their exploitation.
The report containing confidential information can be used internally by the customer, or it can disclose publicly after all vulnerabilities are fixed – upon the decision of the customer.

Scope and Code Revision Date

Initial Audit Repository	https://github.com/hord/farming-geyser/blob/single-sided-farm/contracts/
Initial Audit Files	SingleSidedFarm.sol
Initial Audit Commit	4e6b857a89e246866231394b80a5d78f347586e5
Initial Audit Date	11.08.2021
Secondary Audit Repository	https://github.com/Tokensfarm/tokensfarm-contracts/tree/master/contracts
Secondary Audit Files	TokensFarm.sol
Secondary Audit Commit	325ac367a92cb11c2da3816ce236856458d04e53
Secondary Audit Date	31.08.2021

Table of contents

Introduction

This report presents the findings of the security assessment of Customer` s smart contract and its code review conducted between August 7th 2021 – August 11th 2021; secondary audit was conducted between August 27th 2021 – August 31st 2021.

Scope

The scope of the project is Hord TokensFarm.sol smart contract, which can be found via link:
https://github.com/Tokensfarm/tokensfarm-contracts/blob/master/contracts/TokensFarm.sol
We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the widely known vulnerabilities that considered (the full list includes them but is not limited to them):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Compiler version not fixed
- Unchecked external call – Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary

According to the assessment, Hord smart contracts security risk is medium; some high issues were found for the smart contract during initial audit. These issues were addressed before secondary audit, however, 1 new medium issue was introduced. There are a few medium and low issues leftover in the contract.

Our team performed an analysis of code functionality, manual audit and automated checks with Slither and remixed IDE. All issues found during automated investigation manually reviewed and application vulnerabilities presented in the Audit overview section. A general overview presented in the AS-IS section and all encountered matters can be found in the Audit overview section.

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens loss.
High	High-level vulnerabilities are difficult to exploit. However, they also have a significant impact on smart contract execution, e.g. public access to crucial functions.
Medium	Medium-level vulnerabilities are essential to fix; however, they can't lead to tokens loss.
Low	Low-level vulnerabilities are mostly related to outdated or unused code snippets.
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can generally be ignored.

AS-IS overview

SingleSidedFarm.sol

SingleSidedFarm is a smart contract for ERC20 token farming.
Contract TokensFarm is Ownable, ReentrancyGuard.
SingleSidedFarm has following parameters, enums and structs:

- enum EarlyWithdrawPenalty that has values NO_PENALTY, BURN_REWARDS, REDISTRIBUTE_REWARD
- struct UserInfo that stores uint256 amount; uint256 rewardDebt, unit depositTime.
- IERC20 tokenStaked;
- uint256 lastRewardTime;
- uint256 accERC20PerShare;
- uint256 totalDeposits;
- bool public isEarlyWithdrawAllowed;
- uint256 public minTimeToStake;
- IERC20 public erc20;
- uint256 public paidOut;
- uint256 public rewardPerSecond;
- uint256 public totalRewards;
- mapping (address => bool) public isTokenAdded;
- PoolInfo public pool;
- mapping (address => StakeInfo[]) public stakeInfo;
- uint256 public startBlock;
- uint256 public endBlock;
- EarlyWithdrawPenalty public penalty;
- uint256 fundCounter;
- address public congressAddress;

SingleSidedFarm contract has following functions and modifiers:

- validateStakeByStakeId – modifier that checks whether stake exists
- constructor – public function that sets contract parameters
- setMinTimeToStake – external function that sets minimum stake time. Has onlyOwner modifier.

- _setEarlyWithdrawPenalty – internal function that sets withdrawal penalty
- fund – public function that adds farms fund and calls internal fundInternal function
- _fundInternal – internal function that increases totalRewards and endblock
- _addPool – internal function that sets new pool info
- deposited – external view function that returns user's amount deposited to specific pool. Has validateStakeByStakeId modifier
- pending – external view function that returns user's pending farmed amount for specific stake. Has validateStakeByStakeId modifier
- depositTimestamp – public view function that returns the deposit timestamp. Has validateStakeByStakeId modifier
- totalPending – external view function that returns total pending amount for all stakes
- updatePool – public function that updates reward variables for pool
- deposit – public function to deposit tokens to farm
- withdraw – public function to withdraw tokens from farm. Has validateStakeByStakeId modifier
- emergencyWithdraw – public function that withdraws tokens without rewards farmed. Has validateStakeByStakeId modifier
- getNumberOfUserStakes – external view function that returns number of user stakes
- getUserStakesAndPendingAmounts – external view function that returns user stakes and pending amounts
- getTotalRewardsLockedUnlocked – external view function that returns locked/unlocked rewards
- _erc20Transfer – internal function that transfer ERC20 token to specified address

Audit overview

Only the initial smart contract code and implemented bug fixes were reviewed, no other additions to the code were examined.

Critical

No critical issues were found.

High

1. [Fixed] Users can escape from funds lock via calling emergencyWithdraw function when early withdrawal is forbidden. In this case he would lose all rewards, however, he would instantly return his stake. It's recommended to allow emergency withdraw only after the minimum stake period is passed.
2. [Fixed] setEarlyWithdrawPenalty should be called only once. setEarlyWithdrawPenalty and addPoll are only functions with onlyOwner modifier, thus, it's recommended to move their logic to constructor and remove Ownable functionality.

Medium

3. After the second call of the fund function, ownership is transferred to congress. However, the fund function is external and anyone could call it 2 times just after the contract is deployed, thus, ownership will be transferred.

Low

4. No license is specified for the contract. It's recommended to specify license type before deploying to mainnet.
5. [Fixed] isTokenAdded mapping is declared, however, never used within the contract. It's recommended to remove it.
6. [Fixed] As far as the contract is for a single token farm, there is no need to use a struct for PoolInfo, it could be regular parameters, which would utilize less gas.
7. [Fixed] Line 213 condition could be moved to line 218, what will save a little amount of gas during execution.
8. If a contract is not funded before the start block, it will be blocked in its state. Consider funding it in the constructor.
9. [Fixed] addPool function shouldn't be able to updatePool.

Lowest / Code style / Best Practice

10. [Fixed] Contract comments are saying that the contract is for LP tokens, however, any ERC20 token could be used for staking.

11. [Fixed] fundInternal, erc20Transfer are internal functions, it's recommended to start internal function names with _.

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. For the contract, high-level description of functionality presented in As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security engineers found several high issues during the initial audit. They were fixed before secondary audit, however, one medium issue appeared. There is one leftover medium issue in the smart contract code.

Disclaimer

The smart contracts given for audit were analyzed following the best industry practices at the date of this report, concerning: cybersecurity vulnerabilities and issues in smart contract source code, the details of it were disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

The audit doesn't make any warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the system, bug free status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is essential to note that you should not rely on this report only. We recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee specific security of the audited smart contracts.