CYBER UNIT
Cyber Security Strategic Partner
www.cyberunit.tech

# Smart Contract Audit

Date: April 23, 2021
Report for: Hord
By: CyberUnit.Tech

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

| | |
|---|---|
| Repository | https://github.com/hord/farming-geyser/ |
| Files | Farm.sol |
| Commit | 4ffcb88066468181510881a0ce6aa48b5e66ba11 |
| Date | 23.04.2021 |

# Table of contents

## Introduction

This report presents the findings of the security assessment of Customer`s smart contract and its code review conducted between April 16th 2021 – April 23rd 2021.

## Scope

The scope of the project is Hord smart contracts, which can be found in repo:
https://github.com/hord/farming-geyser/
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 low; no critical, high issues were found for the smart 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

## Farm.sol

Farm is a smart contract for ERC20 token farming.

Contract Farm is Ownable

Farm has following parameters and structs:

- struct UserInfo that stores uint256 amount; uint256 rewardDebt;
- struct PoolInfo that stores IERC20 lpToken; uint256 allocPoint; uint256 lastRewardBlock; uint256 accERC20PerShare;
- IERC20 public erc20;
- uint256 public paidOut = 0;
- uint256 public rewardPerBlock;
- uint256 public totalRewards;
- PoolInfo[] public poolInfo;
- mapping (uint256 => mapping (address => UserInfo)) public userInfo;
- uint256 public totalAllocPoint = 0;
- uint256 public startBlock;
- uint256 public endBlock;

Farm contract has following functions:

- constructor – public function that sets contract parameters
- poolLength – external view function that return total number of pools
- fund – public function that adds farms fund and increases end block
- add – public function that adds new LP to the pool. Has onlyOwner modifier
- set – public function that changes pool's allocation points. Has onlyOwner modifier
- deposited – external view function that returns user's amount deposited to specific pool
- pending – external view function that returns user's pending farmed amount for specific pool
- totalPending – external view function that returns total pending farmed amount
- massUpdatePools – public function that calls updatePool for each pool
- updatePool – public function that updates reward variables for specified pool
- deposit – public function to deposit LP tokens to farm
- withdraw – public function to withdraw LP tokens from farm
- emergencyWithdraw – public function that withdraws LP tokens without rewards farmed
- erc20Transfer – internal function that transfer ERC20 token to specified address

# Audit overview

## Critical

No critical issues were found.

## High

No high issues were found.

## Medium

1. Add doesn't check for the same lp token being in the poolInfo. It's recommended to track all LP tokens so they couldn't be pooled 2 times.

## Low

2. Most public functions can be changed to external ones which will save some gas for the execution.
3. paidOut and totalAllocPoint are initialized with 0 value, it takes more gas for deployment rather to declare them.
4. massUpdatePools is a very expensive function in terms of gas. Increasing the number of pools may make it uncallable due to gas limit and it can be potentially DoSed.
5. There is potential token leftover due to integer division in the fund function.
6. EnumerableSet is imported, however, never used within the codebase.
7. safeTransfer is not used for erc20 transfer in erc20Transfer function

## Lowest / Code style / Best Practice

8. add function naming is confusing. SafeMath Library also has an add function. It's recommended to rename it to addPool.
9. Pools can't be removed. Consider adding functionality to remove pools.

# 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 medium, low and best practice issues that don't have direct security impact.

## Disclaimer

The smart contracts given for audit have 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 which disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

The audit doesn't make 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.