SECURITY AUDIT OF

DEFIHORSE STAKING SMART CONTRACT

Public Report

Apr 13, 2022

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Apr 13, 2022. We would like to thank the DeFiHorse for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the DeFiHorse Staking Smart Contract. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified one vulnerable issue in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About DeFiHorse Staking Smart Contract

The DeFiHorse is a metaverse e-sport game based on Blockchain technology and NFTs, empowering the players and creators to the next level of the horse race.

The game brings you the gorgeous legendary horses to enter the limitless cyberpunk horserace. Overwhelming yourself in every minute of the battle, using your personal and teamwork skills to earn token rewards, and you will end up with a 1,000,000 dollar prize pool.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the DeFiHorse Staking Smart Contract.

It was conducted on commit 0025a48b1a16d9ef297169fcf130a03480e45605 from git repository *https://github.com/DefiHorse/staking/*.

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The DeFiHorse Staking Smart Contract was written in Solidity language, with the required version to be ^0.8.0.

2.1.1. DefiHorseStaking contract

This is the staking contract in the DeFiHorse Staking Smart Contract which extends Ownable and Pausable. With Ownable, by default, Contract Owner is contract deployer, but he can transfer ownership to another address at any time. Token Owner can pause/unpause contract using Pausable contract, user can only stake tokens when contract is not paused.

Users can stake tokens into the pool to get reward tokens. Users may withdraw staked tokens after the endTime. The reward may be released after the endTime + lockRewardDuration.

2.2. Findings

During the audit process, the audit team found one vulnerability issues in the given version of DeFiHorse Staking Smart Contract.

2.2.1. Users can claim rewards multi times CRITICAL

In the _calcReward function, info.reward only be assigned to zero for the memory variable so users can claim rewards multi times.

```
function _calcReward(bytes32 _value)
        internal
        view
        returns(uint256 claimableReward)
    {
        userInfoStaking memory info = infoStaking[_value];
        uint256 releaseTime = info.endTime + LOCK_REWARD_DURATION;
        if(block.timestamp < releaseTime) return 0;
        claimableReward = info.reward;
        info.reward = 0;
    }
```

RECOMMENDATION

Remove view for _calcReward function and change info variable from memory to storage.

```
function _calcReward(bytes32 _value)
        internal
        returns(uint256 claimableReward)
    {
        userInfoStaking storage info = infoStaking[_value];
        uint256 releaseTime = info.endTime + LOCK_REWARD_DURATION;
        if(block.timestamp < releaseTime) return 0;
        claimableReward = info.reward;
        info.reward = 0;
    }
```

UPDATES

- *Apr 13, 2022*: This issue has been acknowledged and fixed by the DeFiHorse team.

2.3. Additional notes and recommendations

2.3.1. Best-practice coding style INFORMATIVE

Some variables in the contract are uppercase which may cause misunderstanding that they are constants.

```
uint256 public END_TIME_EVENT;
    uint256 public LOCK_REWARD_DURATION = (10 minutes);
    uint256 public TOTAL_STAKED = 0;
    uint256 public TOTAL_CLAIMED = 0;
```

UPDATES

- *Apr 13, 2022*: This issue has been acknowledged and fixed by the DeFiHorse team.

2.3.2. Missing check array length INFORMATIVE

The setOptionsStaking function uses some input arrays but missing check the length of these arrays. If the length of _optionInfoDay is larger than other arrays, the contract may access out of bound of the arrays which cause an error and revert.

```
function setOptionsStaking(uint256[] memory _optionInfoDay, uint256[] mem…
  ory _optionInfoAPY,uint256[] memory _optionInfoMaxPool) public onlyOwne…
  r{
        for(uint256 i=0; i < _optionInfoDay.length; i++){
            OptionsStaking memory info = OptionsStaking(_optionInfoDay[i]…
  , _optionInfoAPY[i], _optionInfoMaxPool[i], 0);
            infoOptions[i] = info;
        }
    }
```

UPDATES

- *Apr 13, 2022*: This issue has been acknowledged and fixed by the DeFiHorse team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Apr 13, 2022*	Public Report	Verichains Lab

Table 2. Report versions history