SECURITY AUDIT OF

HOLDSTATION LAUNCHPAD

Public Report

Oct 12, 2023

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.
BSC	Binance Smart Chain or BSC is an innovative solution for introducing interoperability and programmability on Binance Chain.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Oct 12, 2023. We would like to thank the Holdstation 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 Holdstation Launchpad. 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 found some vulnerabilities in the application.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Holdstation Launchpad

Holdstation Defutures is a decentralized leveraged trading platform that combines liquidity efficiency, robustness, and user-friendliness.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the smart contracts of Holdstation Launchpad.

It was conducted on commit `63e3392bc02d9c4101038a083dd1eb2fed0945af` from git repository link: *https://gitlab.com/hspublic/hs-launchpad*.

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
`2a6be26e957a24e0d84b23168a62f948e7b06e58781c9af53b7fde69ecb94962`	`PrivateSale.sol`
`753ca1433b315237ab3a27e49f85b9cf43c655b5750a270442827a8406ecb418`	`TransferHelper.sol`

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 were 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 Holdstation Launchpad was written in `Solidity` language. The source code was written based on OpenZeppelin's library.

2.1.1. HoldStation Private Sale contract

This is the private sale contract in the Holdstation Launchpad which extends `OwnableUpgradeable`, `Initializable`, and `ReentrancyGuardUpgradeable`. With `OwnableUpgradeable`, the Contract Owner is initially set as the initialize caller but has the ability to transfer ownership to another address at any time.

Users have the option to deposit either the native token or specified tokens into the contract in order to receive "earning" tokens. The reward calculation is based on both the number of tokens staked and the allocation provided by the owner. Users are allowed to withdraw their earning tokens and refund tokens after the calculation. Additionally, a small number of tokens will be distributed to the referrer as an incentive.

Note: Sine the contract is upgradable, the contract `owner` can upgrade them with the logic that is not in our audit scope.

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of Holdstation Launchpad.

#	Issue	Severity	Status
1	Missing transfer tokens when staking	CRITICAL	Fixed
2	SpendAmount concern	LOW	Acknowledged
3	Gas optimize	INFORMATIVE	Fixed

Table 2. Finding List

2.2.1. Missing transfer tokens when staking CRITICAL

The `stake` function boosts the user's staked balance; however, its logic does not include the transfer of tokens to the contract. Consequently, anyone can invoke the stake function to increase their staked balance.

```
function stake(uint _amount, address _referer) external isActive nonReentrant {
    require(address(stakingToken) != address(0), "STAKE_ERC20_IS_NOT_ACTIVE");
    require(_amount > minimumStakingAmount, "UNDER_MINIMUM");
    _stake(_amount, _referer);
}

function _stake(uint _amount, address _referer) private {
    address stakerAddress = _msgSender();
    stakers[stakerAddress] += _amount;
    totalStaked += _amount;
    address refererAddress = address(0);
    if (_referer != stakerAddress) {
        RefererDetail storage refInfo = refererDetail[stakerAddress];
        if (refInfo.referer == address(0) && _referer != address(0)) {
            refInfo.referer = _referer;
            refInfo.amount = _amount;
        } else if (refInfo.referer != address(0)) {
            refInfo.amount += _amount;
        }
        refererAddress = refInfo.referer;
    }

    if (!_isStakerAddressExist(stakerAddress)) {
        _addStakerAddress(stakerAddress);
    }
    emit Staked(_msgSender(), _amount, totalStaked, refererAddress);
}
```

UPDATES

- *Oct 06, 2023*: This issue has been acknowledged and fixed by the Holdstation team.

2.2.2. SpendAmount concern LOW

Currently, the calculation of the `spendAmount` relies on the `hardCap` and `allocation` states, but the logic is not clearly defined for these states in the normal way. Some errors may occur when the operator is confused between these states. To improve accuracy, it is hoped that when the client provides us with the document.

```
function getRefundingAmount(address _staker) external view returns (uint) {
    uint spendAmount = (userEarn[_staker] * hardCap) / allocation;
    if (!isRefunded[_staker] && stakers[_staker] > spendAmount) {
      return stakers[_staker] - spendAmount;
    }
    return 0;
  }
```

UPDATES

- *Oct 06, 2023*: This issue has been acknowledged by the Holdstation team.

2.2.3. Gas optimize INFORMATIVE

Using `memory` argument will force Solidity to copy it to memory which will cost more gas than using from `calldata` especially when passing large readonly array.

RECOMMENDATION

Consider change `memory` to `calldata` in all contracts for gas saving.

UPDATES

- *Oct 06, 2023*: This issue has been acknowledged and fixed by the Holdstation team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Oct 06, 2023*	Public Report	Verichains Lab
1.1	*Oct 12, 2023*	Public Report	Verichains Lab

Table 3. Report versions history