QuillAudits

Audit Report
February, 2022

For

THE STARSLAB

Contents

Contents

Scope of the Audit

The scope of this audit was to analyze and document the StarLabs smart contract codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20/721 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20/721 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis
Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis
Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption
In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis, Theo.

Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

Risk-level	Description
High	A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.
Medium	The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.
Low	Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.
Informational	These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	0	2	3
Closed	0	0	1	0

Introduction

During the period of 20th Dec 2021 to 27th Dec 2021 - QuillAudits Team performed a security audit for TheStarLabs smart contracts.

The code for the audit was taken from the following official link: https://etherscan.io/address/0x04e30450d731f2cf167ede2cec74f8e6e8e50364#contracts

Fixed In: https://docs.google.com/document/d/1SQGe3IWjj9GfdKMTBpM4zD3jtTlpYPddUbpSxmMOFjY/edit?usp=sharing

Issues Found – Code Review / Manual Testing

High severity issues

No Issues found

Medium severity issues

No Issues found

Low severity issues

1. Outdated Compiler Version (SWC 102)

```
1450
1451   pragma solidity ^0.8.0;
1452
1453   contract TheStarslab is ERC721Enumerable, Ownable {
```

Description

Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version.

Remediation

It is recommended to use a recent version of the Solidity compiler, which is Version 0.8.9

Status: Partially Fixed

2. Costly Loop

Description

The loop in the contract includes state variables like the .length of a non-memory array in the condition of the for loops.
As a result, these state variables consume a lot more extra gas for every iteration of the for a loop.

```
1498   }
1499
1500   function setWhiteList(address[] memory whiteListAddress, bool bEnable) external onlyOwner {
1501       for (uint256 i = 0; i < whiteListAddress.length; i++) {
1502           _mappingWhiteList[whiteListAddress[i]] = bEnable;
1503       }
1504        presaleEnabled = true;
```

The below functions include such loops at the above-mentioned lines:

setWhiteList

Remediation
It's quite effective to use a local variable instead of a state variable like .length in a loop.
For instance,

```
uint256 local_variable = _groupInfo.addresses.length;
 for (uint256 i = 0; i < local_variable; i++) {
          if (_groupInfo.addresses[i] == msg.sender) {
              _isAddressExistInGroup = true;
              _senderIndex = i;
              break;
          }
      }
```

Reading reference link: https://blog.b9lab.com/getting-loopy-with-solidity-1d51794622ad

Status: Fixed

3. Usage of blocktime.stamp

Line no. 1527, 1529, 1553, 1568, 1571, 1572, 1591, 1600

```
1525
1526    function PRICE(address addr) public view returns (uint256) {
1527        require(block.timestamp > _activeDateTime - PRESALE_HOURS , "Mint is not activated");
1528
1529        if (block.timestamp < _activeDateTime) {
1530            if (_presaleEnabled == true){
1531                require(_mappingWhiteList[addr] == true , "Only Whitelist member can mint now");
1532            }
1533            return PRESALE_PRICE;
```

Description
Block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners. Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right, all that is guaranteed is that it is higher than the timestamp of the previous block.

Remediation
You can use an Oracle to get the exact time or verify if a delay of 900 seconds won't impact the logic of the smart contract

Status: Acknowledged

Informational issues

4. Upgradeable contract was missing. Smart contracts deployed using OpenZeppelin Upgrades Plugins can be upgraded to modify their code while preserving their address, state, and balance. This allows you to iteratively add new features to your project, or fix any bugs you may find in production. Link.

Status: Acknowledged

5. Linting issues were found

```
thestarslab.sol
   88:2  error  Line length must be no more than 120 but current length is 129  max-line-length
   89:2  error  Line length must be no more than 120 but current length is 136  max-line-length
  177:2  error  Line length must be no more than 120 but current length is 136  max-line-length
  204:2  error  Line length must be no more than 120 but current length is 122  max-line-length
  293:2  error  Line length must be no more than 120 but current length is 160  max-line-length
  318:2  error  Line length must be no more than 120 but current length is 156  max-line-length
  515:2  error  Line length must be no more than 120 but current length is 132  max-line-length
  876:2  error  Line length must be no more than 120 but current length is 136  max-line-length
  934:2  error  Line length must be no more than 120 but current length is 136  max-line-length
 1520:2  error  Line length must be no more than 120 but current length is 124  max-line-length
 1575:2  error  Line length must be no more than 120 but current length is 155  max-line-length

✖ 11 problems (11 errors, 0 warnings)
```

Status: Acknowledged

6. Missing test cases

Test cases for the code and functions have not been provided

Remedy
It is recommended to write test cases for all the functions. Any existing tests that fail must be resolved. Tests will help determine if the code is working in the expected way. Unit tests, functional tests and integration tests should have been performed to achieve good test coverage across the entire codebase.

Status: Acknowledged

Functional Tests

We did functional tests for different contracts manually. Below is the report:

function purchase → PASS → Purchase of Token

function airdrop → PASS → airdrop of token

Function setPurchaseLimit → PASS → Setting up of presaleMintLimit, purchaseLimit, publicMintLimit

Function setMintPrice → PASS → set Mint Price

Function setPaymentAddress → PASS → Setting up of Payment Address

Function setPresaleHours → PASS → setting up of Presale Hrs

Results

Few issues were highlighted in the issue section. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines.
3 Low and 3 Informational Issues Found During the Audit, one issue is Fixed and other issues have Been Acknowledged by the Auditee.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of StarLabs. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough; we recommend that the StarLabs Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
February, 2022

For

THE STARSLAB

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com