QuillAudits

Audit Report
July, 2024

For

5TARS

Table of Content

Executive Summary

Project Name	5tars
Overview	One token to rule the 5TARS Ecosystem
Timeline	12th June 2024 - 2nd July 2024
Updated Code Received	25th June
Second Review	26th June 2024 - 29th June 2024
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	The scope of this audit was to analyse the **5tars Contract** for quality, security, and correctness.
Source Code	https://bitbucket.org/valorateam/5tars/src/7a37a1eb94f0294f1219c7097173ccfc7ceb9994
Contracts In-Scope	contracts/FiveStarsOracle.sol contracts/FiveStarsToken.sol
Branch	Main
Contracts Out of Scope	In-scope contracts have been audited by QuillAudits. However, these contracts inherit functionality from out-of-scope Smart contracts that were not audited. Vulnerabilities in unaudited contracts could impact in-scope Smart Contracts functionality. QuillAudits is not responsible for such vulnerabilities. Below are Out of Scope Contracts: • Openzeppelin files
Fixed In	https://bitbucket.org/valorateam/5tars/commits/7c0fad57c03613d1e15b0e75d4458fa3f3b78814

Number of Security Issues per Severity

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	1	1	1

Checked Vulnerabilities

- 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 API violation
- 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 contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC's 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 analyzed 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

A 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 analyzed, 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 behavior 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

Hardhat, Foundry.

Types of Severity

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.

High Severity Issues

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 Severity Issues

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 Severity Issues

Low-level severity issues can cause minor impact and 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 four 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.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

High Severity Issues

No issues were found.

Medium Severity Issues

1. Double accounting will cause some transactions to fail

Path

https://bitbucket.org/valorateam/5tars/src/7a37a1eb94f0294f1219c7097173ccfc7ceb9994/contracts/FiveStarsToken.sol#lines-74

Function

FiveStarsToken.sol::_update()

Description

The update() function in the FiveStarsOracle.sol handles the blacklisting checks, however, also handles the balance limiting checks as well by the _checkWhale() function which read the user's balance and then adds the amount to be transferred when making the check,

```
161             require(
162                 _maxBalanceWhitelist[to] ||
163                     (_tokenContract.balanceOf(to) + amount <=
164                         _maxBalancePerWallet),
165                 "BALANCE_LIMIT"
166             );
```

this logic assumes the user's balance haven't been updated with the transferred amount yet when this check is performed. However, this is not the case in the update() function of the FiveStarsToken.sol contract where the functionality is utilized:

```
69          super._update(from, to, value);
70
71          // Call _tokenOracle.update() if it exists
72
73          if (address(_tokenOracle) != address(0)) {
74              _tokenOracle.update(_msgSender(), from, to, value);
75          }
```

From the above, you'll notice it call the super.update first, which updates the balances with the transferred amount, before making the call to the external update() function in the FiveStarsOracle.sol which then perform the check with adding the amount to be transferred to the balance again even after the balance has already been updated. Due to this, transfer transaction for certain accounts and/or for certain amounts will fail even without reaching the balance limit.

Recommendation

Call the super.update() after the external update checks

Status

Resolved

Low Severity Issues

1. Centralization Risk for trusted owners

Path

FiveStarsOracle.sol, FiveStarsToken.sol

Function

onlyRole()

Description

Contracts have owners with privileged rights to perform admin tasks and need to be trusted to not perform malicious updates or drain funds.

Status

Resolved

Informational Issues

1. No written tests

Path

FiveStarsOracle.sol, FiveStarsToken.sol

Description

It is recommended for protocols to thoroughly test their code using any preferred test suite. However, there are no written tests and as such cannot ascertain the proper functionality of all functions present.

Status

Resolved

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the 5tars codebase. We performed our audit according to the procedure described above.

Some issues of Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.In the End, 5tars team resolved all Issues.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in 5tars smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of 5tars smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the 5tars to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
July, 2024

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com