QuillAudits

Audit Report
November, 2024

For

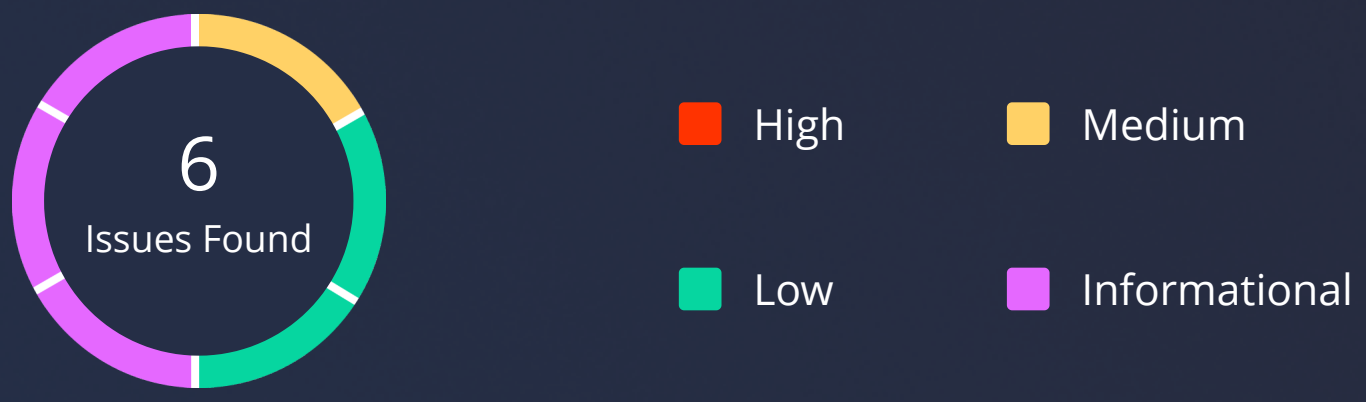
MANGACHAIN

Table of Content

Executive Summary

Project Name	MangaChain Token
Project URL	https://www.mangachain.io/
Overview	Manga Token is a ERC721A NFT contract It implements a tiered minting system with whitelist functionality and profit sharing among owners. The contract has a specific mint date and operates in periods, with different minting rules for different user categories.
Audit Scope	The Scope of the Audit is to Analyse the Security,Code Quality and Correctness of MangaChain Token Contract.
Contracts In-Scope	https://github.com/LeAurelienX/SmartContractMangachain/blob/main/src/MyToken.sol Branch: Main
Commit Hash	007c0cc
Language	Solidity
Blockchain	Ethereum
Method	Manual Analysis, Functional Testing, Automated Testing
First Review	18th November 2024 - 24th November 2024
Updated Code Received	26th November 2024
Second Review	26th November 2024
Fixed In	https://github.com/LeAurelienX/SmartContractMangachain.git Commit : 4318fad “Audit Correction”

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	2	3

Checked Vulnerabilities

- Access Management
- Arbitrary write to storage
- Centralization of control
- Ether theft
- Improper or missing events
- Logical issues and flaws
- Arithmetic Correctness
- Race conditions/front running
- SWC Registry
- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC's conformance
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Multiple Sends
- Using suicide
- Using delegatecall
- Upgradeable safety
- Using throw

Checked Vulnerabilities

- Using inline assembly
- Unsafe type inference
- Style guide violation
- 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 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

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistic analysis.

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.

Medium Severity Issues

1. Anyone in the whitelist is able to call withdraw

Path

MyToken.sol

Function

withdraw()

Description

The withdraw() function intents to withdraw funds for first five owners.
However, isOwners modifier allows all 10 of them to call.
This is not the expected behavior since it allows anyone apart from first five owner to call withdraw

Recommendation

Limit the call so that first 5 owners can call withdraw

Status

Resolved

Low Severity Issues

2. Lack of error message in modifier

Path

MyToken.sol

Function

onlyDeployer()

Description

The onlyDeployer modifier doesn't include an error message, leading to a default "revert" without clarity.

Recommendation

Add a message: require(msg.sender == deployer, "Caller is not the deployer");

Status

Resolved

3. _updateOwnerInfos should have an added safety check

Path

MyToken.sol

Function

_updateOwnerInfos

Description

The _updateOwnerInfos function assumes that the newOwnerInfos array contains exactly 10 elements, but this is not enforced with a require statement.
This can lead to unintended behavior if an incorrect array length is passed during deployment.

Recommendation

Add a
require(newOwnerInfos.length == 10, “Must provide exactly 10 owner infos”);

check in _updateOwnerInfos

Status

Resolved

Informational Issues

4. Redundant Whitelist Entry

Path

MyToken.sol

Function

_setupWhiteLists()

Description

The _setupWhiteLists function has duplicate entries for some addresses (e.g., 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 and 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db .

Recommendation

Remove duplicate addresses.

Status

Resolved

5. Redundant State Variable

Path

MyToken.sol

Description

If the whitelist functionality is hardcoded and there’s no intention to use Merkle trees, remove silverMerkleRoot and goldMerkleRoot to simplify the contract. This will save gas cost by removing state variables

Recommendation

Remove merkle root state variables

Status

Resolved

6. Redundant fallback function

Description

Both fallback and receive functions are present but don’t implement distinct behaviors, making one redundant.

Recommendation

Retain only the receive function if no specific behavior is needed for non-ETH calls.

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 MangaChain Token. We performed our audit according to the procedure described above.

Some issues of medium and informational severity were found. Some suggestions, gas optimizations and best practices are also provided in order to improve the code quality and security posture. In the End, MangaChain Team Resolved It.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in MangaChain Token. 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 MangaChain Token. 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 MangaChain Token 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
November, 2024

For

MANGACHAIN

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com