QuillAudits

Audit Report
April, 2024

For

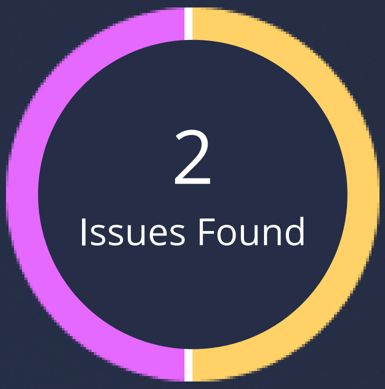
MYSTIC
GAMES

Table of Content

Executive Summary

Project Name	Mystic Games
Overview	Mystic Games is an ERC20 token which will be used as game token.
Timeline	7th May 2024 - 9th May 2024
Updated Code Received	13th May 2024
Second Review	14th May 2024 - 15th May 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 Mystic Game Token Contract for quality, security, and correctness.
Source Code	*https://drive.google.com/file/d/1VV-07DaZiqSxns4pXKcUil1_JRAXJRa5/view?usp=share_link*
Contracts In-Scope	Nefta.sol
Branch	NA
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 - owanable.sol Openzeppelin - draft-ERC20Permit.sol
Fixed In	*https://drive.google.com/file/d/1gt7JB14dU7QaC2mHfTxS64f6kMePunvr/view?usp=share_link*

Number of Security Issues per Severity



🟥 High 🟨 Medium

🟩 Low 🟪 Informational

	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	0	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.

Note: All the tokens will be minted to the owner when deployed

High Severity Issues

No issues were found.

Medium Severity Issues

1. Unlimited minting of token supply

Path

NeftaERC20.sol

Function

Mint()

Description

In contract NeftaERC20.sol, which is token contract has minting functionality to mint the tokens. According to the team there should be limited supply of tokens but mint function does not have any check which will limit or revert if the supply goes past certain amount. This can be an issue in cases where malicious actor gets control of the owner's wallet and mints more tokens diluting in total token supply.

Proof of Concept

```
function test_Mint() external {
      assertEq(nefta.balanceOf(owner), 100_000_000 * 10 ** 18);

      vm.prank(owner);
      nefta.mint(owner, 100_000_000 * 10 ** 18);
      assertEq(nefta.balanceOf(owner), 200_000_000 * 10 ** 18);
  }
```

Recommendation

Please add the global total_supply variable and check using the require statement that the minting amount isn't going above supply for the same in the mint function.

Status

Resolved

Low Severity Issues

No issues were found.

Informational Issues

2. Unfixed pragma

Description

Nefta.sol does not use a fixed solidity version. The contract is using unfixed pragma (^0.8.8), Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. Using floating pragma does not ensure that the contracts will be deployed with the same version. The most recent compiler version may get selected while deploying a contract with a higher chance of bugs in it.

Recommendation

Remove floating pragma and use a specific compiler version with which contracts have been tested.

Status

Resolved

Functional Tests Cases

- ✓ Should get the name of the token
- ✓ Should get the symbol of the token
- ✓ should get the total supply of the token when deployed
- ✓ should get balance of the owner when contract is deployed
- ✓ should transfer tokens to other address
- ✓ should approve another account to spend token
- ✓ should mint to others address and increase total supply
- ✓ should revert when non-owner calls the mint function
- ✓ Should Transfer and Update the ownership through current Owner

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 Mystic Game codebase. We performed our audit according to the procedure described above.

One issue of Medium and informational severity was found, Some suggestions and best practices are also provided in order to improve the code quality and security posture. In the end, the Mystic game team Resolved both Issues.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Mystic Game 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 Mystic Game 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 Mystic Games to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

1000+
Audits Completed

$30B
Secured

1M
Lines of Code Audited

Follow Our Journey

Audit Report
May, 2024

For

MYSTIC
GAMES

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com