AUDITS

SECURITY ASSESSMENT

MOONRAY

JUNE 15 TH 2022

TABLE OF CONTENTS

LEGAL DISCLAIMER

MH Audits are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts MH Audits to perform a security review.

MH Audits does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

MH Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

The report is provided only for the contract(s) mentioned in the report and does not include any other potential additions and/or contracts deployed by Owner. The report does not provide a review for contract(s), applications and/or operations, that are out of this report scope.

MH Audits' goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

MH Audits represents an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. MH Audits' position is that each company and individual are responsible for their own due diligence and continuous security.

INTRODUCTION

MH Audits is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

Secure your project with MH Audits

We offer field-proven audits with in-depth reporting and a range of suggestions to improve and avoid contract vulnerabilities.

Industry-leading comprehensive and transparent smart contract auditing on all public and private blockchains.

Vunerability checking

A crucial manual inspection carried out to eliminate any code flaws and security loopholes. This is vital to avoid vulnerabilities and exposures incurring costly errors at a later stage.

Contract verification

A thorough and comprehensive review in order to verify the safety of a smart contract and ensure it is ready for launch and built to protect the end-user.

Risk assessment

Analyse the architecture of the blockchain system to evaluate, assess and eliminate probable security breaches. This includes a full assessment of risk and a list of expert suggestions.

In-depth reporting

A truly custom exhaustive report that is transparent and depicts details of any identified threats and vulnerabilities and classifies those by severity.

Fast turnaround

We know that your time is valuable and therefore provide you with the fastest turnaround times in the industry to ensure that both your project and community are at ease.

Best-of-class blockchain engineers

Our engineers combine both experience and knowledge stemming from a large pool of developers at our disposal. We work with some of the brightest minds that have audited countless smart contracts over the last 4 years.

PROJECT SUMMARY

PROJECT INTRODUCTION

Moonray is a next-generation 3rd person action game built in Unreal Engine 5 and Ethereum. Made for true gamers, our passion is PvP combat and large, expansive MMO worlds.

In Moonray you can level up your character, weapons and items and trade them on our ethereum NFT marketplace. Win tokens and NFTs in PvP combat or join up with friends to explore a huge open world environments.

Project Name *Moonray*

Contract Name *MNRY Token*

Contract Address -

Contract Chain *Not Yet Deployed on Mainnet*

Contract Type *Smart Contract*

Platform *EVM*

Language *Solidity*

Codebase *GitHub Repository*

INFO & SOCIALS

Network *Ethereum (ERC20)*

Max Token Supply *1,000,000,000*

Website *https://www.monray.game*

Twitter *https://twitter.com/moonraygame*

Telegram -

Discord *https://discord.gg/moonray*

OpenSea *https://opensea.io/MoonrayGame*

Youtube *https://www.youtube.com/c/MoonrayGame*

AUDIT SCORES

Issues	4
Critical	0
Major	0
Medium	0
Minor	3
Informational	1
Discussion	0

All issues are described in further detail on the following pages.

AUDIT SCOPE

FILE	LOCATION
MoonrayToken.sol	*GitHub Repository*
MoonrayTokenBase.sol	*GitHub Repository*
SandwichToken.sol	*GitHub Repository*
SandwichTokenTestHarness.sol	*GitHub Repository*

REVIEW METHODOLOGY

TECHNIQUES

This report has been prepared for Moonray to discover issues and vulnerabilities in the source code of the Moonray project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts producedby industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from major to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective in the comments below.

TIMESTAMP

Version	v1.0
Date	2022/06/08
Description	Layout project Automated / Manual review / Static & dynamic security testing Summary
Version	v1.1
Date	2022/06/15
Description	Reaudit addressed issues Final summary

KEY FINDINGS

TITLE	SEVERITY	STATUS
Floating Pragma	Minor	*Fixed*
Pragma Version Too Recent	Minor	*Fixed*
Large Number Literals	Minor	*Fixed*
Hardcoded Static Address	Informational	*Acknowledged*

Description: *Locking the pragma helps ensure that the contracts do not accidentally get deployed using an older version of the Solidity compiler affected by vulnerabilities.*

The contracts found in the repository were allowing floating or unlocked pragma to be used, i.e., ^0.8.11*. This allows the contracts to be compiled with all the solidity compiler versions above* 0.8.11*.*

Location: *contracts/MoonrayToken.sol L02*
contracts/MoonrayTokenBase.sol L02
contracts/SandwichToken.sol L02
contracts/SandwichTokenTestHarness.sol L02

Impacts: *If the smart contract gets compiled and deployed with an older or too recent version of the solidity compiler, there's a chance that it may get compromised due to the bugs present in the older versions or unidentified exploits in the new versions.*

Incompatibility issues may also arise if the contract code does not support features in other compiler versions, therefore, breaking the logic. The likelihood of exploitation is really low therefore this is only informational.

Issue: *Floating Pragma*

Type: *Floating Pragma (SWC-103)*

Level: *Minor*

Recommendation: *Keep the compiler versions consistent in all the smart contract files. Do not allow floating pragmas anywhere.*

Reference: https://swcregistry.io/docs/SWC-103

Alleviation: *The Moonray team opted to consider our references and applied the recommendation.*

Description:

Solidity constantly releases new compiler versions and it is not recommended to stay on the most latest and recent versions as there may be unidentified bugs, inconsistencies, and exploits present in the newer versions.

Location: *contracts/MoonrayToken.sol L02*
contracts/MoonrayTokenBase.sol L02
contracts/SandwichToken.sol L02
contracts/SandwichTokenTestHarness.sol L02

Impacts:

Recent compiler versions are not time-tested and may be susceptible to unknown vulnerabilities and exploits.

Issue: *Pragma Version Too Recent*

Type: *Missing Best Practices*

Level: *Minor*

Recommendation: *It is suggested to use a compiler version that is neither too recent nor too old. A stable compiler version should be used that is time-tested by the community, which fixed vulnerabilities introduced in older compiler versions. The code should be kept updated according to the compiler release cycle. It should be tested before going on the mainnet to reduce the chances of new vulnerabilities being introduced.*

It is recommended to use version 0.8.7 *which is the most stable at this time.*

Alleviation: *The Moonray team opted to consider our recommendation and applied the suggested* 0.8.7 *complier version.*

Description:

Solidity supports multiple rational and integer literals, including decimal fractions and scientific notations. The use of very large numbers with too many digits was detected in the code that could have been optimized using a different notation also supported by Solidity.

Location: *contracts/MoonrayToken.sol L14*
contracts/SandwichToken.sol L14

```
    constructor()       MoonrayTokenBase('MoonrayToken',       'MNRY',
1000000000, TOKEN_RECEIVER) {
      // Implementation version: 1
   }
...

...

        constructor()    MoonrayTokenBase('SandwichToken',    'SNDWCH',
1000000000, TOKEN_RECEIVER) {
      // Implementation version: 1
   }
```

Issue: *Large Number Literals*

Type: *Gas Optimisation*

Level: *Minor*

Recommendation: *Scientific notation in the form of* 2e10 *is also supported, where the mantissa can be fractional but the exponent has to be an integer. The literal* MeE *is equivalent to* M * 10**E. *Examples include* 2e10, 2e10, 2e-10, 2.5e1*, as suggested in official solidity documentation*

https://docs.soliditylang.org/en/latest/types.html#rational-and-integer-literals

It is recommended to use “ 1e9***”***

Alleviation: : *The Moonray team heeded our references and applied the suggested recommendation.*

Impacts:

Having a large number literals in the code increases the gas usage of the contract while its deployment and when the functions are used or called from the contract. It also makes the code harder to read and audit and increases the chances of introducing code errors.

Description:

The contracts was found to be using hardcoded addresses on L12*. A private address constant variable "*TOKEN_RECEIVER*" was defined which was using hardcoded addresses for the token receivers wallet. This could have been optimized using dynamic address update techniques along with proper access control to aid in address upgrade at a later stage.*

Location: *contracts/MoonrayToken.sol L12*
contracts/SandwichToken.sol L12

```
address private constant TOKEN_RECEIVER =
0x5B38Da6a701c568545dCfcB03FcB875f56beddC4; // TODO: Fill in Token
Receiver.

...

...

address private constant TOKEN_RECEIVER =
0x05Ed4cf991c4ed7606930AB54dDbF27836C1f590; // Testnet Wallet
```

Issue: *Hardcoded Static Address*

Type: *Missing Best Practices*

Level: *Informational*

Recommendation: *It is recommended to create dynamic functions to address upgrades so that it becomes easier for developers to make changes at a later stage if necessary. The said function should have proper access controls to make sure only administrators can call that function using access control modifiers. There should also be a zero address validation in the function to make sure the tokens are not lost.*

Alleviation: *The Moonray team responded with the following transcript:*

The static address is only used in our constructor method, and we do not want that to be editable. Since the only mint is the initial Token Generation Event, there will never be a future reason for the address to be used.

Impacts:

Hardcoding address variables in the contract make it difficult for it to be modified at a later stage in the contract as everything will need to be deployed again at a different address if there's a code upgrade.

Private GitHub Repository

FINDING CATEGORIES

The assessment process will utilize a mixture of static analysis, dynamic analysis, in-depth manual review and/or other security techniques.

This report has been prepared for Moonray project using the above techniques to examine and discover vulnerabilities and safe coding practices in Moonray's smart contract including the libraries used by the contract that are not officially recognized.

A comprehensive static and dynamic analysis has been performed on the solidity code in order to find vulnerabilities ranging from minor gas optimizations to major vulnerabilities leading to the loss of funds.

Various common and uncommon attack vectors will be investigated to ensure that the smart contracts are secure from malicious actors. The testing methods find and flag issues related to gas optimizations that help in reducing the overall gas cost It scans and evaluates the codebase against industry best practices and standards to ensure compliance It makes sure that the officially recognized libraries used in the code are secure and up to date.

AUDIT SCORES

MH Audits AuditScores is not a live dynamic score. It is a fixed value determined at the time of the report issuance date.

MH Audits AuditScores are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports and scores are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts MH Audits to perform a security review.

WEBSITE
MHAUDITS.IO

TWITTER
@MHAUDITS