QuillAudits

Audit Report
March, 2022

For

METAGAMEHUB

Contents

Scope of Audit 01

Check Vulnerabilities 01

Techniques and Methods 02

Issue Categories 03

Number of security issues per severity. 03

Introduction 04

High Severity Issues 05

1. Insufficient require checks on the createPool parameters 05

Medium Severity Issues 05

2. For loop over dynamic array leads to out of gas 05

3. Staking Token is assumed to be non-malicious 06

4. Insufficient require checks on cycletoken in updatePool() 06

5. Insufficient liquidity in the Staking Pool 07

Low Severity Issues 08

6. Missing error messages 08

7. Missing zero address check 08

8. Insufficient Documentation Provided 09

Informational Issues 09

9. There is an unused function parameter 09

Functional Tests 10

Automated Tests 11

Closing Summary 12

Scope of the Audit

The scope of this audit was to analyze and document the Metagamehub 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 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 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	0	0
Closed	1	4	3	1

Introduction

Between **Jan 14, 2022 - Jan 25.2022** - QuillAudits Team performed a security audit for TheDac smart contracts.

The code for the audit was taken from following the official link:

https://github.com/metagamehub/Tokenomics-Contracts/blob/main/contracts/Staking/LockedStakingRewards.sol

Fixed In: https://github.com/metagamehub/Tokenomics-Contracts/tree/audit-fix/contracts/Staking

V	Date	Commit ID	Files
1	Jan 14	f3899d0a31dbb7386eacb7efefbff2ac32ce7cbb	contracts/*
2	Jan 24	94e2027d9638a64fd1fc95f094975fd6f3b6013b	contracts/*

Issues Found – Code Review / Manual Testing

High severity issues

1. Insufficient require checks on the createPool parameters

Line	Code

87	for (i; i < inputsLen; i++) {
	lockLen = _input[i].unlockAt.length;
	for (ii; ii < lockLen; ii++) {
	if (_input[i].account == address(0)) {
	require(false, "Zero address");
	} else if (
	_input[i].unlockAt.length != _input[i].amounts.length ||
	_input[i].unlockAt.length > MAX_LOCK_LENGTH
	) {
	require(false, "Wrong array length");
	} else if (_input[i].unlockAt.length == 0) {
	require(false, "Zero array length");
	}

Description

The parameters tokenPerShareMultiplier, cycleDuration, startOfDeposit and tokenPerShare in the constructor and createPool() function do not have any required checks on their range of values allowed and thus, can be exploited by the owner.

Remediation

It is advised to add proper “require” checks for the above mentioned variables.

Status: Fixed

Medium severity issues

2. For loop over dynamic array leads to out of gas

Line	Code

45	for (uint256 i = 0; i < _initialPools.length; i++) {
	createPool(i, _initialPools[i]);
	}

Description
There is a for loop used over dynamic array _initialPools in the constructor.

Remediation
It is advised to add a “require” check on the length of this array as it can lead to out of gas issues.

Status: Fixed

3. **Staking Token is assumed to be non-malicious**

Description
In the latest commit that contains comments, it is added- "We don’t use Reentrancy Guard here because we only call the stakeToken contract which is assumed to be non-malicious". But there are no proofs given to prove this.

Remediation
It is advised that the team provide sufficient details such as any audits done for this token. Perhaps such assumptions can be dangerous to assume.

Status: Fixed
Auditor’s comment: The staking token is the MGH token which is a MiniMeToken. The token was audited and its audit report can be found here, https://docs.google.com/document/d/1oxLfheb_w27BtBHVU4J_2rb4KK8kSagnQMifGJkkwkQ/edit?usp=sharing

4. **Insufficient require checks on cycletoken in updatePool()**

Line	Code
70	pool[_pool].startOfDeposit += pool[_pool].cycleDuration;

Description
cycleDuration in the updatePool() function has no checks and thus, startOfDeposit could be updated by a very large or a very small value which may lead to unwanted or undiscovered outcomes.

Remediation
It is advised to add proper “require” checks for the above mentioned variable.

Status: Fixed

5. Insufficient liquidity in the Staking Pool

Line	Code
-	Across the code

Description
If a user deposits say 50,000 Staking tokens and the owner updates the tokenPerShareMultiplier to 15000, then it is expected that the user will get 50% more tokens than he initially deposited.
But this scenario can fail if enough staking tokens are not available in the contract. That is it is expected that the pool has 25,000 additional tokens (75,0000 in total).

It is also possible that users A, B and C each deposit 50,000 tokens and then after updating the pool (with tokenPerShareMultiplier set to 15000), user A withdraws 75,000, user B withdraws 75,000 and then the user C is not able to withdraw even a single token.

Remediation
Thus it is required to add sufficient checks and methods to see that the users do get the promised returns.

Status: Fixed
Clients’s comment: Sufficient liquidity has been supplied to staking contract

Low severity issues

6. Missing Error messages

Line	Code
51	require(stakeToken.transferFrom(_sender, address(this), _amount));
62	require(stakeToken.transfer(msg.sender, _tokenAmount));

Description
There is missing error message in the require statement on lines: 51 and 62

Remediation
It is advised to add appropriate error messages.

Status: Fixed

7. Missing zero address check

Line	Code
36	function receiveApproval (address _sender, uint256 _amount, address _stakeToken, bytes memory data)

Description
Missing zero address check for _sender parameter in the receiveApproval() function.

Remediation
Added a “require” check for checking the zero address for the sender parameter.

Status: Fixed

8. **Insufficient documentation provided**

Description
There is insufficient documentation provided as to the business logic of the project and the expected implementation of the staking pools.

Remediation
It is advised to provide more documentation for the same.

Status: Fixed

Informational issues

9. There is an unused function parameter _stakeToken in the receiveApproval() function. It is advised to make clear as to why this parameter is unused and to remove it if not required.

Status: Fixed

Functional Tests

- [x] should be able to create a new pool and set cycleDuration of locking period
- [x] should able to receive approval, transfer and stake tokens
- [x] should be able to update the sharesAmount, unstake and withdraw tokens
- [x] should be able to transfer ownership to the production wallet
- [x] should be able to update pool and revert if a pool is terminated
- [x] should be able to updateTokenPerShareMultiplier when the pool is in transferPhase
- [x] should be able to terminate Pool and return the value of the Transfer Phase as well.
- [x] should be able to return sharesToToken/tokenToShares amount and Pool’s Information
- [x] should be able to view the user's token amount.
- [x] should revert if cycleDuration is zero or more.
- [x] should revert if the withdraw function is called and the pool is locked.
- [x] should revert if the user tries to deposit 0 tokens.
- [x] should revert if the user tries to update a terminated pool.
- [x] should revert if the start of deposit time is not greater than the block’s timestamp.
- [x] should revert if the reward amount is negative.

Automated Tests

Slither

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

Mythril

No issues were reported by Mythril.

Closing Summary

No instances of Integer Overflow and Underflow vulnerabilities are found in the contract.

Numerous issues were discovered during the initial audit. All issues are Fixed by the Auditee.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Metagamehub.

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 Metagamehub Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
March, 2022

For

METAGAMEHUB

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com