QuillAudits

Audit Report
May, 2022

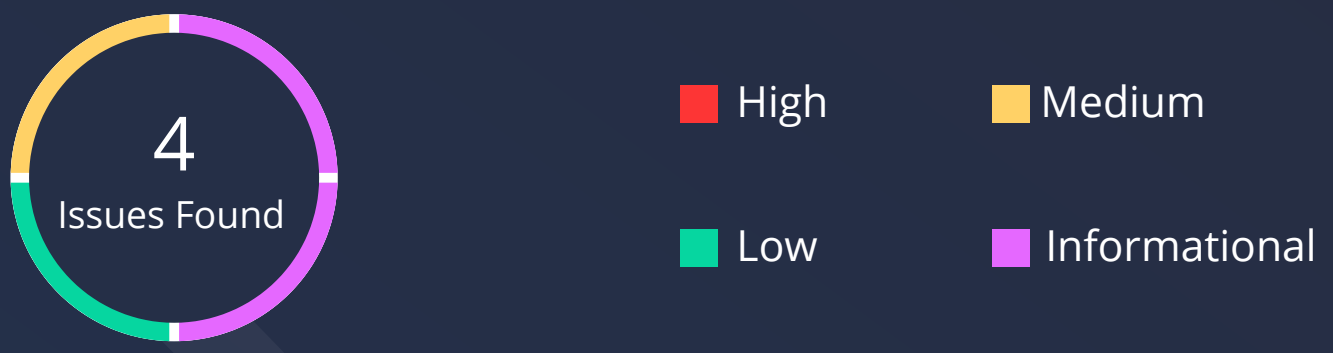
For

BEYOND IMAGINATION TECHNOLOGIES

Table of Content

Executive Summary

Project Name	Beimagine
Timeline	May 19th, 2022 to May 25th, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse Beimagine codebase for quality, security, and correctness.
Sourcecode	*https://github.com/Quillhash/Project71-LOYALTY/blob/main/loyalty%20contract.sol*
Fixed in	*https://github.com/Quillhash/Project71-LOYALTY/blob/main/loyalty3.sol*



	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	2

Types of Severities

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.

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.

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
- 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 leve

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.

Manual Analysis

A. Contract - LoyaltyContract

High Severity Issues

No issues were found

Medium Severity Issues

A.1 Expired points are not Deducted

Line #57, 132

```
function allocatePoints(string memory _userIdString, uint256 _points, uint256 _expiryDate,
    require (totalSupply + _points <= maxSupply, "Maximum points limit exceeded.");
    require(_expiryDate > block.timestamp, "Expiry date should not be in the past.");
    bytes32 _userId = keccak256(abi.encodePacked(_userIdString));
```

Description

According to the contract's logic, the supply of maximum points are capped (limited). Although, there is a check that the totalSupply+_points (allocated or modified) must be less than or equal to maxSuppy but there is no removal of points after they are expired. However, if the expired points are not removed from the contract then the contract will neither be able to allocate any new points nor modify the existing ones. Hence, the contract would be inoperable

Remediation

To solve the issue, we would recommend to use a function that can be called to remove the expired points.

Status

Fixed

Low Severity Issues

A.2 Centralization Risk

Description

Using the “OnlyOwner” modifier means that only the owner of the contract will be able to call the functions regarding allocation of new points, modification of points, redeem points, etc. It poses two risks, first one is in a scenario where the owner will lose their Private Key then in that case the contract will be frozen, will never be able to work again, and all the points could be lost because all the externally callable functions contain the “OnlyOwner” modifier. Secondly, the working can be very inefficient as only the user with the Owner’s private will be able to call the functions.

Remediation

Consider adding a factory contract to be the Owner of this contract then let the users interact with that contract.

Status

Fixed

Informational Issues

A.3 Unlocked pragma (pragma solidity ^0.8.0)

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Recommendation

Here all the in-scope contracts have an unlocked pragma, it is recommended to lock the same. Moreover, we strongly suggest not to use experimental Solidity features (e.g., pragma experimental ABIEncoderV2) or third-party unaudited libraries. If necessary, refactor the current code base to only use stable features.

Status

Fixed

A.4 General Recommendations for Gas Optimization

In conclusion, we would like to mention that some of the public functions that are never called from within the contract should be declared external in order to save gas.

- setOwner
- allocatePoints
- redeemUserPoints
- modifyCouponPoints

Status

Fixed

Functional Testing

Some of the tests performed are mentioned below

- Should be able to allocate points
- Should be able to redeem points
- Should be able to modify points
- Should be able to get points available to a user
- Should be able to update points
- Should be able to transfer ownership
- Should revert if total supply of points exceeds maxSupply
- Should revert if a user tries to redeem after the expiry date
- Should revert if the user doesn't have enough points

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 Beimagine. 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, Beimagine Team Resolved all issues.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Beimagine Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

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

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.

500+
Audits Completed

$15B
Secured

500K
Lines of Code Audited

Follow Our Journey

Audit Report
May, 2022

For

BEYOND IMAGINATION TECHNOLOGIES

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com