QuillAudits

Audit Report
December, 2023

For

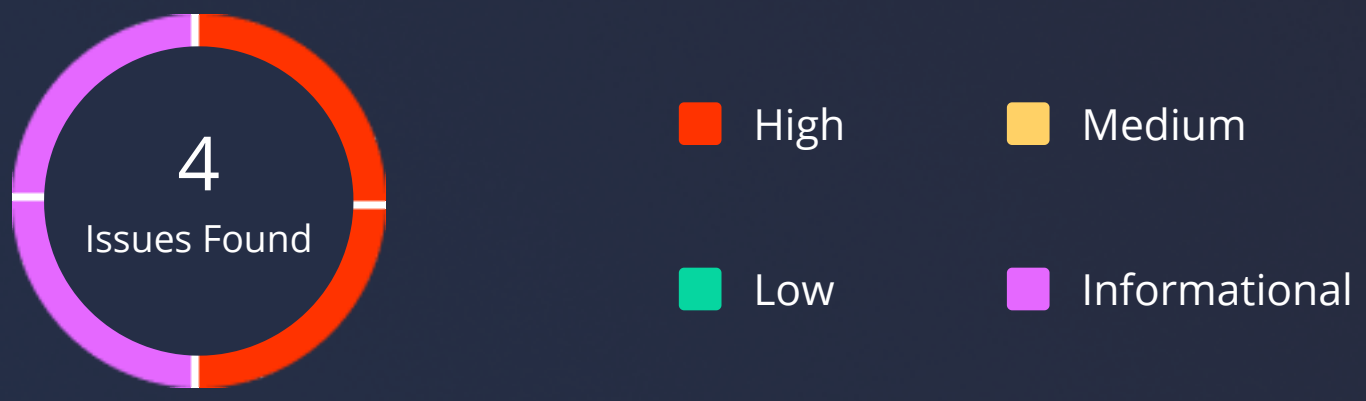
HURALYA

Table of Content

Executive Summary

Project Name	Huralya
Overview	LYA is the primary and native token used for all Huralya products, which includes the Genesis game and NFT market. The majority of transactions, fees, and related activities within the Genesis game will necessitate the use of LYA. It starts with a total supply of 100,000,000 LYA for the initial four years. Beyond this, only another 100 millions will be mintable in the second four years. So finally, the max supply is 200 millions.
Timeline	19th December 2023 - 22nd December 2023
Updated Code Received	22nd December 2023
Second Review	22nd December 2023
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 analyze the Huralya codebase for quality, security, and correctness.
Source Code	*https://github.com/huralya/contract/blob/main/contracts/Huralya.sol*
Branch	Main
Fixed In	*561a394*

Number of Security Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	2
Partially Resolved Issues	0	0	0	0
Resolved Issues	2	0	0	0

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 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-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 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 impacts 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.

High Severity Issues

1. Incorrect calculation at getMintableAmount()

Path

https://github.com/huralya/contract/blob/0a4c60e871628b3d2ee4c862ecf1f86ab4400a72/contracts/Huralya.sol

Function

getMintableAmount

Description

getMintableAmount() function is incorrectly calculating **currentMintableAmount**. This function is supposed to return 100 million tokens every 4 years, but it returns fewer than expected tokens when more time passes than 4 years.

Also, when the owner mints funds after 12 years then the function returns the right amount. But when the owner again mints after 3 years(i.e. 15 years in total) then it returns more amount than it should return which will lead to the minting of more funds than even after 16 years.

Test

https://gist.github.com/Makg2/b8f9511e69cc47b5b6c942a9a4cf1c29

Note

The Huralya team intends to use the contract for about 4 years only, that's why they changed the implementation at hash 087f427f279f14fd402d59b6420cf0f70aabaf6a.

Status

Resolved

2. The owner can mint more than 200 million tokens after 4 years

Path

https://github.com/huralya/contract/blob/087f427f279f14fd402d59b6420cf0f70aabaf6a/contracts/Huralya.sol

Function

mint

Description

mint() function only checks if totalSupply < MAX_SUPPLY. After 4 years owner will mint, and totalSupply will be 200 million tokens. But if someone burns their tokens then totalSupply will decrease, and the owner will again be able to mint 100 million tokens.

Test

https://gist.github.com/Makg2/14809209589f53b88e752d4c3c63792e

Note

Huralya team changed the implementation to remove the bug at hash 561a3944ee75b52aa33c45e65825163e445eb604.

Status

Resolved

Informational Issues

1. Missing _amount > 0 check

Path

https://github.com/huralya/contract/blob/561a3944ee75b52aa33c45e65825163e445eb604/contracts/Huralya.sol

Function

mint

Description

mint() function allows 0 amount minting. It could lead to phishing.

Recommendation

Add a check to check _amount > 0.

Status

Acknowledged

2. MINT_TIMESTAMP can be declared as immutable

Path

https://github.com/huralya/contract/blob/561a3944ee75b52aa33c45e65825163e445eb604/contracts/Huralya.sol

Function

MINT_TIMESTAMP

Description

MINT_TIMESTAMP is only updated in the constructor. So, it can be declared as immutable to save gas in **mint()** function.

Recommendation

uint256 public immutable MINT_TIMESTAMP;

Status

Acknowledged

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

Some issues of High and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Huralya 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 Huralya 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 Huralya Team 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.

850+
Audits Completed

$30B
Secured

$30B
Lines of Code Audited

Follow Our Journey

Audit Report
December, 2023

For

HURALYA

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com