QuillAudits

Audit Report
July, 2022

For

NUGEN COIN

Table of Content

Executive Summary

Project Name	Nugen Operational
Overview	Nugen Operational smart contract is a distribution ecosystem that allows an array of qualified distribution wallets to claim Nugen BEP20 tokens after specific claim periods. This contract is designed to make a contract owner add addresses and revoke. Users can retrieve their reward information and also claim tokens.
Timeline	11 July, 2022 - 14 July, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Source Code	*https://github.com/nugenuniverse/nugencoin/blob/main/Operational.sol*
BSC Mainnet Address	0x17DBaB3D0c053120598Da6F12B4Cc50ce05EED7E

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	0	**3**	**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
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

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 BEP-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 Testing

A. Contract - Nugen Operational

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

1. Missing Check for Zero Address

addDistributionWallet(address[ ] memory account, uint[ ] memory amount, uint[ ] memory startTime)

Description
Function receives an array of addresses as one of the parameters to add and set an address qualified to claim Nugen tokens. In conditions where the addresses are numerous and the owner is unaware of the address zero being in the list, the contract will assign tokens to this address.

Remediation
Zero-address check is important to prevent token allocation that is not claimable.

Status

Resolved

2. Missing Events Emission

Description
Event emissions are relevant to help track smart contract state changes. When important actions are carried out, whether for the owner to add or revoke addresses, events must be emitted and indexed. This will enable and ease the processes of querying onchain data of this smart contract.

Remediation
Event should be emitted in the revokeDistributionWallet that allows the owner to revoke addresses.

Status
Resolved

3. Use a Constructor in place of Initializable Contract

Description
The contract uses the initializable contract from Openzeppelin to help initialize variables. While this is an old contract with a floating solidity version, “>= 0.4.24 < 0.7.0”, it is unfit to be integrated with a contract with version 0.8.11 because this could pose security lapses present in the old version. It is also recommended for Initializable contract to be used for upgradable contracts but this contract is not upgradable. The constructor function is invoked only once before a contract is deployed, which makes variables set at the constructor level immutable.

Remediation
Use a constructor to initialize critical parameters for the contract and the inherited Initializable contract should be removed.

Status
Resolved

Informational Issues

4. Lengthy and Unclear Error Message

Line	revokeDistributionWallet(), claim()
289, 290	`require(user_.totalClaimed < user_.balance, "Operational : total claim < total balance");` `require((user_.lastClaim > 0) && ((user_.lastClaim + claimPeriod) <= block.timestamp), "Operational : lastClaim < block.timestamp");`

Description

Some error messages have symbols that do not explain clearly why a function call fails when it does not meet some requirement. Some errors are lengthy and facilitate higher gas consumption.

Remediation

Brief error messages without a symbol are recommended to help understand the reasons for function call reverts.

Status

Resolved

5. Absence of Code Comments

Description

Contract lacks appropriate code commenting that explains the motive behind the contract creation. Structured code have attributes of good code commenting that details the operation a function is created for and aids easy understanding of the contract when interacting with it.

Remediation

It is recommended for the contract codebase to be properly commented. A Natspec comment format is advisable. This allows users and developers interacting with the contract to understand the motive behind the contract.

Status

Resolved

6. General Recommendation

For variable initialization, it is recommended that a constructor is used in replacement of the Initializable contract inherited in the contract. A constructor helps the contract set the variables at run time and avoid possible bugs that could be in the old initializable library.

In our audit we have concluded that the ownership is crucial in the operational contract, so we would still recommend using a two way process to transfer the ownership.

A check for zero addresses will prevent the addition of the address zero as one of the qualified addresses for the operation token. It is recommended therefore that a zero check is added to the addDistributionWallet function.

Good code comments are also important to aid quick code understanding. When users and developers first come across the code without proper comments, it will take a long period to understand the contract. The Natspec comment format is recommended.

addDistributionWallet() should be a two step process to eliminate frontrunning in the scenario where: Admin adds distribution for the address which is ineligible for the reward and admin tries to revoke the amount added for distribution using revokeDistributionWallet(). In this case a user can frontrun this transaction and can claim the reward in some cases before revoking it.

Note: The Nugen team has acknowledged the risk of frontrunning. However, Implementing a two-step process will increase gas cost and that's why the Nugen team decided to move ahead with the existing mechanism for adding distributions, looking at the low possibility of this scenario.

Functional Tests

- Should initialize the contract and get the token address
- Should revert on calling the initialize function twice
- Should get the subsequentClaim
- Should get the claimPeriod
- Should set the subsequentClaim of the contract
- Should set the claimPeriod of the contract
- Should revert on adding distribution wallets with mismatch length
- Should get reward balance of users not added to the reward pool
- Should revert if user cannot claim from reward pool
- Should add addresses when owner calls addDistributionWallet

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 Nugen Operational smart contract. We performed our audit according to the procedure described above.

Some issues of Low 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 audit is not a security warranty, investment advice, or an endorsement of the Nugen 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 Nugen Team 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
July, 2022

For

NUGEN COIN

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com