QuillAudits

# Audit Report
July, 2022

For

NUGEN COIN

# Table of Content

# Executive Summary

| | |
|---|---|
| **Project Name** | Nugen Reserve |
| **Overview** | NUGEN is the indigenous coin of the NUGEN platform. It is a BEP-20 standard token built on the robust Binance Smart Chain network. NUGEN will serve as a collateralized token that facilitates cryptocurrency payments on the network. |
| **Timeline** | 8 July, 2022 - 13 July, 2022 |
| **Method** | Manual Review, Functional Testing, Automated Testing etc. |
| **Audit Scope** | *https://testnet.bscscan.com/address/0x6a33DC0Bb2F7Ded14e0E5d604Bc601cA9761df4D#code* |
| **Fixed In** | *https://gitlab.com/stsblockchain/neugen-smart-contract/-/blob/main/Reserve.sol* |
| **Commit Hash** | 3c5bd9c6da8b9feff4eb4eea409e0bdf0248cf3c |
| **BSC Mainnet Address** | 0x567ef1048C14AEEAdFC8a71c048491B1A6aFab18 |

5
Issues Found
🟥 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 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 Testing

## A. Contract - Nugen Reserve

### High Severity Issues

No issues found

### Medium Severity Issues

No issues found

### Low Severity Issues

#### 1. Missing Events

**Description**
Missing events don't pose any security risk. However, it makes it difficult to track on chain events especially in decentralized environments of reserves and other operations that revolve around communities. It is advisable to emit an event whenever a significant action takes place on a contract.

**Remediation**
Consider adding events to the following functions:
- revokeDistributionWallet
- setTotalNumberOfVesting
- setcliff
- setSubClaimPerMonth
- setClaimPeriod

**Status**
**Resolved**

## 2. Missing zero check

**Function - addDistributionWallet()**

**Description**
There should be a zero address check in this function because there will be a large number of accounts being passed as the parameters. Although it’s an onlyOwner function, in this case an array of addresses is involved which poses more risk of passing a zero address as a distribution wallet, and the process will end up burning the tokens allocated.
If this is the intended behavior then please comment below.

**Remediation**
Consider adding a zero check in the function

**Status**

**Resolved**

## 3. Use of old libraries

**Description**
There is an old library being used in the contract from lines “#151 to #197” by Openzeppelin named “Initializable.sol” that specifies the pragma version as “>=0.4.24 <0.7.0” which is outdated and this contract relies on the initializable library.
It is not recommended to use libraries/contracts intended for old versions of solidity because they may pose a security risk.

**Remediation**
Consider using the latest initializable by Openzeppelin.

**Status**

**Resolved**

## Informational Issues

### 4. Commented code

**Description**
There are some instances of code being commented out in the following files that should be removed

**Remediation**
Remove the commented code, or address them properly.

**Status**

**Resolved**

### 5. Missing zero check for amount

**Function - setSubClaimPerMonth()**

**Description**
There should be a zero amount check in this function because if it is set to zero then in a scenario in the condition on line #323 where the calculation(line 327) depends on the sub claim amount to be greater than zero otherwise the function will revert on line 334.

**Remediation**
Consider adding a zero amount check in the function

**Status**

**Resolved**

## 6. General Recommendation

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

We would also like to point out that the error messages described in the contract are not very clear or explanatory if seen from the user’s point of view. Hence, they are rather technical. We would recommend to put more clear and explanatory error messages so that the users will be informed well about the problem with their claim of the reserves.

There should be a zero check for the amount passed in the “addDistributionWallet” function so the address won’t get allocated 0 tokens by mistake and takes up the space in the array for no reason.

We would also recommend using a constructor instead of an initializer function if the contract is not upgradable.

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

- ✓ Owner Should be able to initialize the contract only once
- ✓ Owner should be able to allocate a distribution wallet
- ✓ Owner should be able to revoke any wallet
- ✓ Owner should be able to set subsequent claim period
- ✓ Owner should be able to set subsequent claim amount
- ✓ Owner should be able to call failcase whenever necessary
- ✓ Owner should be able to set total number for vesting
- ✓ The owner should be able to set the cliff.
- ✓ Users should be able to call the claim function for the token reserves
- ✓ Should revert if the claim function is called before the start time of the contract
- ✓ Should revert if the accounts and the amount array's lengths don't match
- ✓ Should revert if the balance of the contract is less than the amount requested
- ✓ Should revert if the transfer fails in case of failcase

# 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 Token Reserve. 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.
In the end, Nugen Team resolved all Issues.

# 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