QuillAudits

Audit Report
July, 2022

For

NUGEN COIN

Table of Content

Executive Summary

Project Name	Nugen Reward Pool
Overview	Nugen Reward Pool is a smart contract designed to reward specific addresses that are added by the owner, alongside each address reward amount and the corresponding time for each address to claim. Users can claim their individual rewards and retrieve reward information.
Timeline	8 July, 2022 - 13 July, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Audit Scope	*https://testnet.bscscan.com/address/0xbb41a7f89aa8937bd5ffbc9b84134043efa592d0#code*
Fixed In	*https://gitlab.com/stsblockchain/neugen-smart-contract/-/blob/main/RewardPool.sol*
Commit Hash	40bee3d5f1e63f9f66acc840e8899c638903ca29
BSC Mainnet Address	0xCBdb6C60265F2216Cf34950A1bc02c0a7eFc8D7a

6
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**	**3**

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	✓ Tautology or contradiction
✓ Timestamp Dependence	✓ Return values of low-level calls
✓ Gas Limit and Loops	✓ Missing Zero Address Validation
✓ Exception Disorder	✓ Private modifier
✓ Gasless Send	✓ Revert/require functions
✓ Use of tx.origin	✓ Using block.timestamp
✓ Compiler version not fixed	✓ Multiple Sends
✓ Address hardcoded	✓ Using SHA3
✓ Divide before multiply	✓ Using suicide
✓ Integer overflow/underflow	✓ Using throw
✓ Dangerous strict equalities	✓ 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 Reward Pool

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

1. Missing Zero Address Validation

Description
In this contract, the owner has the ability to add an array of addresses that are qualified to claim rewards from the contract. However, when these addresses are added, there is a missing check for the zero address. Allocated tokens assigned to this address could facilitate the burning of tokens.

Remediation
There should be a zero address check when the owner is adding an array of distribution wallets to claim the reward.

Status

Resolved

2. Missing Events Emission

Description
Emitted events from smart contracts ease the processes of tracking relevant information on the blockchain. While some actions were emitted in the contract, a critical function that allows the owner to revoke distribution wallets was exempted. This action also needs to be emitted to aid in tracking revoked addresses.

Remediation
The revokeDistributionWallet should have an event that emits a revoked address.

Status

Resolved

3. No constructor in the contract

Description
Contracts with no constructor allows for the constant change of state variables which doubts the immutability of these variables. Even though there is an integration of the initializable contract from Openzeppelin, which could be called just once, having a constructor gives a high level of security and allows for setting of variables at run time before deployment. This better affirms to users that the state variables won't be later changed to their detriment.

Remediation
Use the constructor to set variables at run time since the contract is not an upgradable contract.

Status

Resolved

Informational Issues

4. Use of Old Libraries

Description
The initializable library integrated into the contract is outdated and reads a solidity pragma version, “>= 0.4.24 < 0.7.0”. Outdated versions have bugs that have been fixed in the latest version and it is advisable to use the latest versions of the library to escape any possible risks that are inherent in these old versions.

Remediation
Consider using the newest version of libraries in the contract.

Status

Resolved

5. Absence of Comments

Description
Contract lacks appropriate comments that explain the logic behind every function. Comments, particularly the Natspec format, are important to explain the functions, parameters passed into functions and the motive. Good commented codes give contracts finesse and better structure.

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. Lengthy and Unclear Error Message

Line	revokeDistributionWallet(), claim()
265	`require(account.length <2000,"RewardPool : length <2000");`
269	`require(user_.totalClaimed < user_.balance, "RewardPool : user claimed all funds or user may not be added");`

283, 284	`require(user_.totalClaimed < user_.balance, "RewardPool : total claim < total balance");`
	`require((user_.lastClaim > 0) && ((user_.lastClaim + claimPeriod) <= block.timestamp), "RewardPool : lastClaim < block.timestamp");`

Description

Lengthy string error messages consume more gas when these functions are called. While it is imperative to have error messages that hint at a problem, it is recommended for them to be brief and clear. Like the error message in line 265 and 283 which should explain the failure to meet a condition, in the contract, the error messages have symbols that don't better convey the failure for a function call.

Remediation

Error messages should be brief and concise. It should make the user understand reasons for the failure of a function call.

Status

Resolved

7. General Recommendation

Zero address checks are important checks that must not be evaded. Being an address which is not owned by anybody, it is important to check that it doesn’t become one of the qualified addresses added by the owner when the “addDistributionWallets” function is triggered.

Inherited contracts and libraries in the reward pool contract are products of the Openzeppelin standard, however, some of the libraries are outdated and it is recommended to settle for the latest versions of the libraries. In the case of the ownable contract, it is advisable that a two-step process is adopted to ensure safe ownership transfer.

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

Changing state variables in a for-loop consumes more gas and it is recommended to create a local variable for the purpose of a loop and outside the loop, the state variable is afterwards updated or assigned the values that a local variable holds. This is one technique to escape hiking gas during a function call with for loops.

Non-technical and concise errors are helpful to convey reasons for the failure of function calls. However, it is recommended for them to be short to save gas. Alongside this, appropriate comments are also relevant to structure codebases for easy comprehension for users.

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 Reward Pool. 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+	**$15B**	**500K**
Audits Completed	Secured	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