QuillAudits

AUDIT REPORT

April , 2025

For

GRQ

GET RICH QUICK INSTITUTE

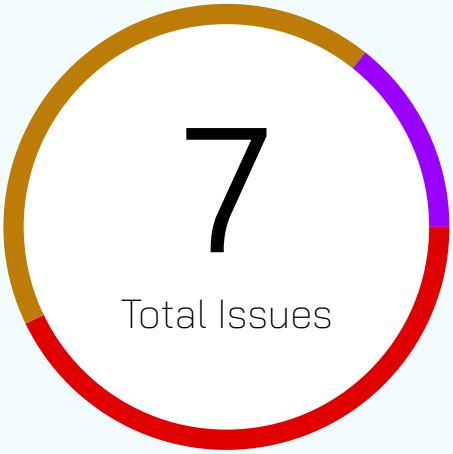
Table of Content

Executive Summary

Project name	GRQ
Project URL	https://docs.grq.institute/
Overview	GenesisRewardPool smart contract implements a short-term (7-day) staking mechanism to bootstrap the ini tial supply of a token. Users can deposit tokens into various pools to earn rewards, with each pool having configurable al location points and deposit fees. The contract includes fea tures for single or batch deposits/withdrawals, emergency withdrawals, and a claim-all function to collect rewards from multiple pools at once. The operator role has administrative privileges to add/modify pools and recover funds
Audit Scope	The scope of this Audit was to analyze the GRQ Smart Contracts for quality, security, and correctness.
Source Code link	https://github.com/grq-institute/core-contracts/com-mit/2d889f43cc77aa3f053e48a6c09c358acde0bb3c
Contracts in Scope	src/farms/ShareRewardPool.sol src/helpers/TokenVault.sol src/helpers/OracleV2.sol src/core/Masonry.sol src/core/Treasury.sol src/tokens/PEG.sol src/tokens/SHARE.sol src/tokens/BOND.sol
Branch	Master
Commit Hash	2d889f43cc77aa3f053e48a6c09c358acde0bb3c
Language	Solidity

Blockchain	Berachain
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	16th April 2025 to 24th April 2025
Updated Code Received	GRQ team parallely fixed the issues as we found.
Review 2	23rd april 2025 to 24th apriil 2025
Fixed In	29b9716e4b6e3ed5c9d4d35de0404b1aaa474624

Number of Issues per Severity



High	3 (42.86%)
Medium	3 (42.86%)
Low	0 (0.00%)
Informational	1 (14.29%)

Severity

Issues	High	Medium	Low	Informational
Open	0	0	0	0
Resolved	**3**	**3**	0	0
Acknowledged	0	0	0	**1**
Partially Resolved	0	0	0	0

Checked Vulnerabilities

- ✔ Access Management
- ✔ Arbitrary write to storage
- ✔ Centralization of control
- ✔ Ether theft
- ✔ Improper or missing events
- ✔ Logical issues and flaws
- ✔ Arithmetic Computations Correctness
- ✔ Race conditions/front running
- ✔ SWC Registry
- ✔ Re-entrancy
- ✔ Timestamp Dependence
- ✔ Gas Limit and Loops
- ✔ Exception Disorder
- ✔ Gasless Send
- ✔ Use of tx.origin
- ✔ Malicious libraries
- ✔ Compiler version not fixed
- ✔ Address hardcoded
- ✔ Divide before multiply
- ✔ Integer overflow/underflow
- ✔ ERC's conformance
- ✔ Dangerous strict equalities
- ✔ Tautology or contradiction
- ✔ Return values of low-level calls
- ✔ Missing Zero Address Validation
- ✔ Private modifier
- ✔ Revert/require functions
- ✔ Multiple Sends
- ✔ Using suicide
- ✔ Using delegatecall
- ✔ Upgradeable safety
- ✔ Using throw

- [x] Using inline assembly
- [x] Unsafe type inference
- [x] Style guide violation
- [x] 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 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

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistical analysis.

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 impact 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 Issues

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

Security vulnerabilities identified that must be resolved and are currently unresolved.

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

Incorrect Reward Debt Calculation in ShareReward-Pool's Deposit Function Leads to Excess Rewards.

Resolved

Path

ShareRewardPool.sol

Function

deposit()

Description

In the deposit function of ShareRewardPool.sol, there is an issue that leads to users receiving more rewards than they should. The rewardDebt is calculated using the user's old amount before the new deposit amount is added to their balance. This incorrect sequencing results in users being able to claim more rewards than they have actually earned.

Recommendation

The rewardDebt should be calculated after updating the user's amount to ensure it accurately reflects the new deposit.

Underflow in Reward Debt Calculation After Partial Withdrawals

Resolved

Path

ShareRewardPool.sol

Function

withdraw()

Description

The underflow issue in the reward calculation logic when users perform partial withdrawals. The issue occurs when calculating pending rewards after a user has withdrawn most of their tokens but left a small amount. The small remaining amount combined with the accumulated reward per share can cause an arithmetic underflow in the reward debt calculation.

When User performs a partial withdrawal, leaving a very small amount staked The calculation of user.amount * accRewardPerShare / 1e18 becomes smaller than the existing user.rewardDebt

Recommendation

move the rewardDebt calculation line after updating the user's amount.

Missing Pool Total Stake Update in Emergency Withdrawal

Resolved

Path

ShareRewardPool.sol

Function

emergencyWithdraw()

Description

The emergencyWithdraw function in ShareRewardPool contract fails to decrease the pool.totalStaked value when users perform emergency withdrawals. This leads to an inconsistent state where the pool's total staked amount does not reflect the actual staked tokens after emergency withdrawals.
Which impacts incorrect reward calculations due to inflated totalStaked value.

Recommendation

Add the missing state update to decrease the pool's totalStaked amount in the emergencyWithdraw function.

Medium Severity Issues

Missing Fee Status Validation in Pool Addition Function

Resolved

Path

ShareRewardPool.sol

Function

add()

Description

The add function in ShareRewardPool contract lacks proper validation for fee configuration when adding new pools. While the contract correctly handles zero fee cases in the set function by disabling fee collection, this validation is missing in the add.

Recommendation

Add the missing validation check in the add function

Denial of Service in claim() Function Due to Failed Fee Recipient Transfers

Resolved

Path

ShareRewardPool.sol

Function

claim()

Description

The claim() function in ShareRewardPool contract is vulnerable to Denial of Service (DoS) when fee transfers fail. Currently, if a fee transfer to the recipient fails, the entire claim transaction reverts due to a hard requirement (require(success)). This means if the fee recipient is unable to receive native tokens for any reason (contract without receive function, out of gas, etc.), users will be completely blocked from claiming their earned rewards.

Recommendation

Implement a graceful failure handling that disables fees for the pool if transfer fails.

Incorrect Comparison Operator in _safeRewardTransfer

Resolved

Path

ShareRewardPool.sol

Function

_safeRewardTransfer()

Description

The _safeRewardTransfer function in ShareRewardPool contract uses an incorrect comparison operator (< instead of <=) when validating reward amounts against available rewards. This prevents users from claiming the exact amount of available rewards

Recommendation

Change the comparison operator from < to <=

Informational Severity Issues

Unused functions and redundant code

Acknowledged

Description

1. The setLockUp function in the Masonry contract contains a redundant validation check for _withdrawLockupEpochs and _rewardLockupEpochs. The same condition is checked twice.

2. Unused code/functions in Treasury.sol

isInitialized(), getPegUpdatedPrice(),getReserve()

```
        require(
            _withdrawLockupEpochs > 0 && _rewardLockupEpochs > 0 && _claimRewardsBurnEpochs > 0,
            "lockupEpochs must be greater than 0"
        );
        require(_withdrawLockupEpochs > 0 && _rewardLockupEpochs > 0, "lockupEpochs must be > 0");//@audit redundant check
```

Recommendation

remove them.

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 GRQ. We performed our audit according to the procedure described above.

Issues of High,Medium and Low severities were found , GRQ team resolved most of them and acknowledged the one.

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

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

7+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

April , 2025

For

GRQ
GET RICH QUICK
INSTITUTE

QuillAudits

Canada, India, Singapore, UAE, UK
www.quillaudits.com audits@quillaudits.com