QuillAudits

Audit Report
September, 2024

For

Torque

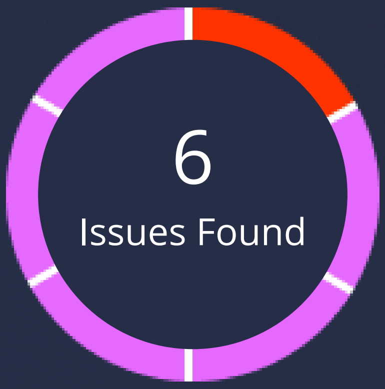
Table of Content

Executive Summary

Project Name	Torque
Project URL	https://www.torque.fi/
Overview	The protocol facilitates the refinancing of crypto assets between Aave and Radiant lending pools. It allows users to repay loans using USDC or USDCe and withdraw equivalent amounts of WBTC or WETH. The protocol is designed to optimize loan management, offering flexible refinancing options while maintaining secure and efficient asset transfers.
Audit Scope	https://arbiscan.io/address/0x6b5c6C6a5223e1Ce30d24dedE62848bAd0845Fd2 https://arbiscan.io/address/0xB4C189f3697F21703AbbB6550e97D8dDF4c3DF3B#code https://arbiscan.io/address/0xa4C46c137Be22E8fb6033509041917C70665fD24#code https://arbiscan.io/address/0x1dBAEc382a74743504607FFe73aD22F9142396D0#code
Contracts in Scope	- RadiantWbtcRefinance - RadiantWethRefinanceUSDC - AaveWbtcRefinance - AaveWethRefinance
Commit Hash	NA
Language	Solidity
Blockchain	Arbitrum

Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	2nd September 2024
Updated Code Received	3rd September 2024
Review 2	3rd September 2024
Fixed In	*https://arbiscan.io/address/0x5c4dc1c9fbc4ddbb29680e1b6ae80432a3942e9b#code* *https://arbiscan.io/address/0x380206e0d634630d1f47ecda0a01e4b95409d9f7#code* *https://arbiscan.io/address/0xeecbd33fa7b52834417fb73ff888cec7c8b1f483#code* *https://arbiscan.io/address/0xe76686e8723e1cf300a43616b5249c144f43d485#code*

Number of Security Issues per Severity



🟥 High	🟨 Medium
🟩 Low	🟪 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	3
Partially Resolved Issues	0	0	0	0
Resolved Issues	1	0	0	2

Checked Vulnerabilities

- ✓ Access Management
- ✓ Arbitrary write to storage
- ✓ Centralization of control
- ✓ Ether theft
- ✓ Improper or missing events
- ✓ Logical issues and flaws
- ✓ Arithmetic 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

Checked Vulnerabilities

✓ Using inline assembly

✓ Unsafe type inference

✓ Style guide violation

✓ 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 Statistic 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

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. Inadequate Handling of Excess Funds in Repay Functions

Function Name

repay

Description

The depositUSDC and depositUSDCe functions currently transfer the full amount of USDC or USDCe specified by the user to the contract, and then call the repay function on the lending protocol. However, if the amount to be repaid is less than the amount sent by the user, the contract does not return the excess funds to the user. This discrepancy can lead to a loss of tokens for the user, as the contract should handle and refund any surplus amount that was not used for repayment.

Recommendation

Update the depositUSDC and depositUSDCe functions to check the actual repay amount returned by the repay function and return any excess tokens to the user.

Status

Resolved

Informational Issues

2. Replace String-Based Error Messages with Custom Errors for Gas Efficiency

Description

The contracts currently use string-based error messages in require statements to handle errors. This approach is less efficient and can increase gas costs due to the on-chain storage of strings. Additionally, string-based errors lack the structured handling that custom errors provide, which can lead to less maintainable code.

For example, the following code uses string-based error messages:

require(usdcAmount > 0, "USDC amount must be greater than 0");
require(rWethAmount > 0, "rWETH amount must be greater than 0");
This practice could be optimized by using custom errors, which are more gas-efficient and provide better structure for error handling.

Recommendation

Refactor the contracts to use custom errors instead of string-based error messages to improve gas efficiency and code clarity.

Status

Acknowledged

3. Missing Zero Address Checks in updateRLendingPool and updateAaveLendingPool

Function Name

updateRLendingPool, updateAaveLendingPool

Description

The updateRLendingPool and updateAaveLendingPool functions in the RadiantWbtcRefinanceUSDC and AaveWbtcRefinance contracts, respectively, lack checks to ensure the new lending pool address is not the zero address. This omission can lead to critical issues if a zero address is set, such as causing contract interactions to fail or resulting in loss of functionality.

Recommendation

Add a zero address check in both functions to revert the transaction if the provided address is the zero address.

Status

Resolved

4. Missing Validation for updateRateMode Parameter

Function Name

updateRLendingPool, updateAaveLendingPool

Description

The updateRateMode functions in the RadiantWbtcRefinanceUSDC, RadiantWethRefinanceUSDC, AaveWbtcRefinance, and AaveWethRefinance contracts do not validate the rateMode parameter. Any uint256 value can be provided, potentially breaking the contract interaction with Aave and Radiant if an invalid rate mode is set.

Recommendation

Add validation in the updateRateMode functions to ensure that only valid rate modes (1 for Stable and 2 for Variable) are accepted.

Status

Acknowledged

5. Missing NATSpec Comments in Functions

Description

The smart contracts lack NATSpec (Ethereum Natural Language Specification) comments for many functions. NATSpec comments are crucial for providing clear documentation on function purposes, parameters, and return values, which helps improve the readability and usability of the contracts. The absence of these comments makes it harder for developers and users to understand the functions' intentions and how they should be used.

Recommendation

Add NATSpec comments to all functions in the smart contracts to clearly describe their purpose, parameters, and return values.

Status

Acknowledged

6. Emergency Withdraw Function May Fail for Certain Tokens

Description

The withdraw function is designed as an emergency mechanism to retrieve funds from the contract. However, it relies on the return value of the transfer function to verify successful transfers. Some tokens, especially non-standard or poorly implemented ones, may not return a value on transfer operations, causing the function to fail and potentially lock the tokens in the contract permanently.

Recommendation

Implement a fallback mechanism or use a different method to ensure funds are withdrawn regardless of the token's transfer behavior, such as using transfer without checking for return values or implementing a different withdrawal strategy.

Status

Resolved

Functional Tests

Some of the tests performed are mentioned below:

- ✓ USDC Deposit: Verify that depositUSDC successfully repays USDC debt and emits the USDCDeposited event.
- ✓ USDCe Deposit: Verify that depositUSDCe successfully repays USDCe debt and emits the USDCeDeposited event.
- ✓ WBTC Withdrawal: Confirm that withdrawWBTC transfers the correct amount of WBTC and emits the WbtcWithdrawn event.
- ✓ ETH Withdrawal: Confirm that withdrawETH transfers the correct amount of ETH and emits the ETHWithdrawn event.
- ✓ Update RLendingPool: Check that updateRLendingPool changes the RLendingPool address and emits the RLendingPoolUpdated event.
- ✓ Update AaveLendingPool: Check that updateAaveLendingPool changes the AaveLendingPool address and emits the AavePoolUpdated event.
- ✓ Rate Mode Update: Verify that updateRateMode changes the rate mode and emits the RateModeUpdated event.

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

Some issues of High/low/medium and informational severity were found. Some suggestions, gas optimizations 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 Torque. 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 Torque. 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 Torque 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 leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
September, 2024

For

Torque

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com