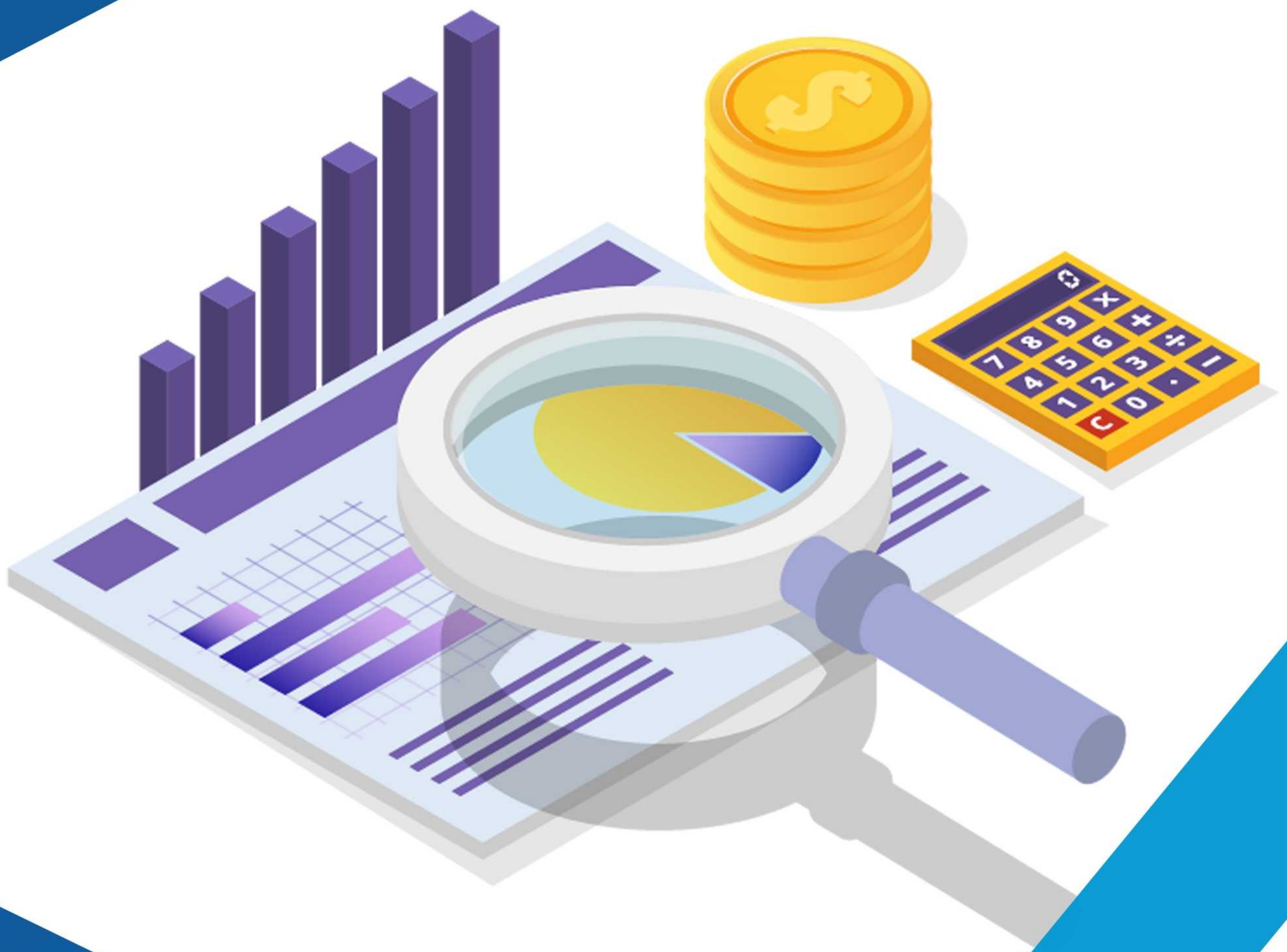
QuillAudits



Audit Report
August, 2021

RELIC

Contents

Scope of Audit

The scope of this audit was to analyze and document the Relic smart contract codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

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 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.
SmartCheck.

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 security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability 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 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 possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

Issue Categories

Every issue in this report has been assigned with 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 to be fixed before moving to a live environment.

Medium level 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 level severity issues

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 four issues which indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	0	3	1
Closed	0	0	3	2

Introduction

From **Aug 2, 2021 to Aug 17, 2021** - QuillAudits Team performed a security audit for Relic smart contracts.

The code for the audit was taken from following the official link: https://docs.google.com/document/d/1IG-cLsGPW8H4s1dIGwjRfEMKecukZfFe0iGJmZFSBBA/edit?usp=sharing

Note	Date	Hash
Version 1	02/08/2021	05D69AE8FACD40CF5373B1F8F36EC11C
Version 2	09/08/2021	b3eff9eb44459096aa5181ed73e7ae6d4cf24a37
Version 3	17/08/2021	https://bscscan.com/address/0x9051398cC35496b532f28418B2D8e0b718FE69DA#code

Issues Found – Code Review / Manual Testing

High severity issues

No issues were found.

Medium severity issues

Low level severity issues

1. Infinite loops possibility at multiple places

Line	Code

446	function includeInReward(address account) external onlyOwnerOrPoolOwner() {
	require(_isExcluded[account], "RELIC: Account is already included");
	for (uint256 i = 0; i < _excluded.length; i++) {
	if (_excluded[i] == account) {
	_excluded[i] = _excluded[_excluded.length - 1];
	_tOwned[account] = 0;
	_isExcluded[account] = false;
	_excluded.pop();
	break;
	}
	}
	}

550	function _getCurrentSupply() private view returns(uint256, uint256) {
	uint256 rSupply = _rTotal;
	uint256 tSupply = _tTotal;
	for (uint256 i = 0; i < _excluded.length; i++) {
	if (_rOwned[_excluded[i]] > rSupply || _tOwned[_excluded[i]] > tSupply) return (_rTotal, _tTotal);
	rSupply = rSupply.sub(_rOwned[_excluded[i]]);
	tSupply = tSupply.sub(_tOwned[_excluded[i]]);
	}
	if (rSupply < _rTotal.div(_tTotal)) return (_rTotal, _tTotal);
	return (rSupply, tSupply);
	}

Description

There are 2 functions in smart contracts, where the array.length variable is used directly in the loop. It is recommended to put some kind of limits.

Functional test

Function Names	Testing results
owner	Passed
poolOwner	Passed
onlyOwner	Passed
onlyOwnerOrPoolOwner	Passed
renounceOwnership	Passed
transferOwnership	Passed
getUnlockTime	Passed
getPreviousOwner	Passed
lockOwnership	Possible to gain ownership after lock period
unlockOwnership	Possible to gain ownership after lock period
_setOwner	Passed
lockTheSwap	Passed
name	Passed
symbol	Passed
decimals	Passed
totalSupply	Passed
balanceOf	Passed
transfer	Passed
transferPoolAccount	Passed
manualSwapAndLiquify	Passed

changePoolOwnership	Passed
updateRouter	Passed
createAMMPair	Passed
setAMMPair	Passed
allowance	Passed
approve	Passed
transferFrom	Passed
increaseAllowance	Passed
decreaseAllowance	Passed
isExcludedFromReward	Passed
totalFees	Passed
excludeFromReward	Critical operation lacks event log
includeInReward	Potential to hit the gas block limit
tokenFromReflection	Passed
reflectionFromToken	Passed
excludeSenderFromFee	Passed
excludeReceiverFromFee	Passed
excludeSenderReceiverFromFee	Passed
setTaxFeePercent	Passed
setNoTokensSellToAddToLiquidity	Passed
setHoldFeePercent	Passed
setLiquidityFeePercent	Passed

setBurnFeePercent	Passed
_burnTokens	Passed
setWhaleTransferLimit	Passed
setWhaleWalletLimit	Passed
setSwapAndLiquifyEnabled	Passed
receive	Passed
_reflectFee	Passed
getCirculatingSupply	Passed
getMaxTransferAmount	Passed
getMaxBalanceAmount	Passed
_getRate	Passed
_getCurrentSupply	Potential to hit the gas block limit
removeAllFee	Passed
transactFeesHoldPool	Passed
restoreAllFee	Passed
isSenderExcludedFromFee	Passed
isReceiverExcludedFromFee	Passed
isSenderReceiverExcludedFromFee	Passed
_approve	Passed
_exceedsMaxBalance	Passed
_exceedsMaxTransactionAmount	Passed
_ChargeFee	Passed

_transferPool	Passed
rescueTokens	Passed
swapBNBForTokensAndAddToPool	Passed
_transfer	Passed
swapAndLiquify	Passed
swapTokensForEth	Passed
addLiquidity	Passed
_tokenTransfer	Passed
_getTValues	Passed
_getRValues	Passed
_getValues	Passed
_transferFromExcluded	Passed
_transferToExcluded	Passed
_transferStandard	Passed
_transferBothExcluded	Passed
FinaliseTransferAndFees	Passed
calculateTaxFee	Passed
calculateHoldFee	Passed
calculateLiquidityFee	Passed
calculateBurnFee	Passed
_takeHoldFee	Passed
_takeLiquidity	Passed

Miscellaneous

Constant/View/Pure functions:

INTERNAL ERROR in module Constant/View/Pure functions: can't convert undefined to object
Pos: not available

Similar variable names:

INTERNAL ERROR in module Similar variable names: can't convert undefined to object
Pos: not available

No return:

INTERNAL ERROR in module No return: can't convert undefined to object
Pos: not available

Guard conditions:

INTERNAL ERROR in module Guard conditions: can't convert undefined to object
Pos: not available

String length:

INTERNAL ERROR in module String length: can't convert undefined to object
Pos: not available

SOLHINT LINTER

```
Relic.sol:248:18: Error: Parse error: missing ';' at '{'
 Relic.sol:261:18: Error: Parse error: missing ';' at '{'
 Relic.sol:273:18: Error: Parse error: missing ';' at '{'
 Relic.sol:290:18: Error: Parse error: missing ';' at '{'
 Relic.sol:302:18: Error: Parse error: missing ';' at '{'
 Relic.sol:398:18: Error: Parse error: missing ';' at '{'
 Relic.sol:421:18: Error: Parse error: missing ';' at '{'
 Relic.sol:447:18: Error: Parse error: missing ';' at '{'
 Relic.sol:539:19: Error: Parse error: extraneous input
'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' expecting ';'

 Relic.sol:856:61: Error: Parse error: missing ';' at '('

 Relic.sol:856:63: Error: Parse error: missing ';' at
'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
```

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the Relic 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 Relic Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines.

No instances of Integer Overflow and Underflow vulnerabilities or Back-Door Entry were found in the contract.

The majority of the concerns addressed above have been acknowledged, implemented, and verified.

RELIC

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com