Chain Integrity

The Web 3.0 Ethical Hacking Company

QUINT

Smart Contract Audit Deliverable

Date: Apr 9, 2022

Version: 1

Table of Contents

03 Overview

1.1 Summary

1.2 Scope

1.3 Documentation

1.4 Review Notes

1.5 Recommendations

1.6 Disclaimer

06 Detailed Analysis

2.1 Severity Ranking

2.2 Observations List

2.3 Observations Review

14 Closing Statement

15 Appendix

A Classification List

B Source Code Fingerprints

1.1 Summary

Project

Name	QUINT
Description	Staking, NFTs, Metaverse, Tokenization
Platform	Binance Smart Chain (BSC)
Codebase	*
Commit	*

Engagement

Delivered	Apr 9, 2022
Methods	Static Analysis, Manual Review, TM, RA.
Consultants	2
Timeline	2 Days

Observations

Total	11	**Status**
Critical	4	Pending
High	1	Pending
Medium	1	Pending
Low	5	Pending
Undetermined	0	

Executive

This document has been prepared for QUINT (Client) to discover and analyze the smart contract provided by the team for security vulnerabilities, code correctness, and risks. The smart contract has been comprehensively examined using structural analysis, behavioral analysis, and manual review techniques.

Throughout the audit, caution was taken to ensure that the smart contract(s):

- Implements robust functions which are safe from well-known attack vectors.
- The logic and behavior adheres to the associated documentation and code comments.
- Transfer flows are designed in a sustainable way, safeguarded from i.e. infinite loops.
- Does not hide potential back doors and implements sanity checks where suitable.

1.2 Scope

File	Fingerprint (SHA-256 Checksum)
QUINT.sol	cb755e35f06a554ec4c02e522700b76a4d3f6f5e78574c52783a8c5d82bad8c3

1.3 Documentation

The smart contract in scope is documented partially but the styling guide and code layout have a quality decreasing effect on the readability and clarity of the code. Next, some of the code comments do not align with the actual implementation causing confusion and incorrectness. Finally, all observations are explicitly based on the information available in the provided codebase and whitepaper.

1.4 Review Notes

Notice the classification type 'Overpowered Design' in the appendix A - this particular type places an increased risk on investors as a result of the design patterns used throughout the codebase.

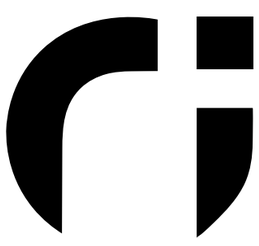
4 observations ranked as critical were found related to broken code or overpowered design patterns. **1** observation ranked as high was found leading to behavioral inconsistencies.

1.5 Recommendations

The codebase in scope should be fixed to conform with the recommendations presented in this assessment. The code should be optimized to follow a specific styling guide, align with the documentation, and the team should assess the sanity checks implemented throughout the code as some result in broken code.

1.6 Disclaimer

It should be noted that this document is not an endorsement of the effectiveness of the smart contracts, rather limited to an assessment of the logic and implementation. This audit should be seen as an informative practice with the intent of raising awareness on the due diligence involved in secure development and make no material statements or guarantees in regards to the operational state of the smart contract(s) post-deployment. Chain Integrity (Consultant) do not undertake responsibility for potential consequences of the deployment or use of the smart contract(s) related to the audit.



For clarity of understanding, observations are arranged from critical to informational. The severity of each issue is evaluated based on the risk of exploitation or other unexpected behavior.

Critical

An issue flagged as critical means that it can affect the smart contract in a way that can cause serious financial implications, catastrophic impact on reputation, or disruption of core functionality.

High

An Issue flagged as high means that it can affect the ability of the smart contract to function in a significant way i.e. lead to broken execution flows or cause financial implicants.

Medium

An Issue flagged as medium means that the risk is relatively small and that the issue can not be exploited to disrupt execution flows or lead to unexpected financial implications.

Undetermined

An issue flagged as undetermined means that the impact of the discovered issue is uncertain and needs to be studied further.

Low

An Issue flagged as informational does not pose an immediate threat to disruption of functionality, however, it should be considered for security best practices or code integrity.

2.2 Observations List

ID	Title	Type	Severity
OBSN-01	**Broken Execution**	Control Flow	⌃⌃
OBSN-02	**Broken Access**	Control Flow	⌃⌃
OBSN-03	**Pontential Theft of Token Pool**	Overpowered Design	⌃⌃
OBSN-04	**Potential LP Honeypot**	Overpowered Design	⌃⌃
OBSN-05	**Code and Documentation Not Matching**	Code Correctness	⌃
OBSN-06	**Reversible State On Function Execution**	Logical Issue	–
OBSN-07	**Explicit Function Mutability**	Code Correctness	⌄
OBSN-08	**Naming Conventions**	Coding Style	⌄
OBSN-09	**Loop Over Unknown Array Size**	Volatile Code	⌄
OBSN-10	**Use of Raw Values**	Magical Numbers	⌄
OBSN-11	**Emitting Events on Sensitive Changes**	Coding Style	⌄

Unchanged **Improved** **Fixed**

OBSN-01

Location: QUINT.sol

Explanation:

The multiSig_Update_2nd_Wallet_ASK(...) and multiSig_Update_2nd_Wallet_CONFIRM(...) functions are not executable because the suggested_New_MultiSig_2nd_Wallet variable has no value and equals the zero address. This prevents both functions from being executed because the accessibility is guarded by a sanity check that ensures that the suggested_New_MultiSig_2nd_Wallet variable must not be the zero address.

Recommendation:

Remove the sanity check that checks whether the suggested_New_MultiSig_2nd_Wallet variable is the zero address, initialize the suggested_New_MultiSig_2nd_Wallet variable in the constructor(), or add a function to set the variable to a value that is different than the zero address.

OBSN-02

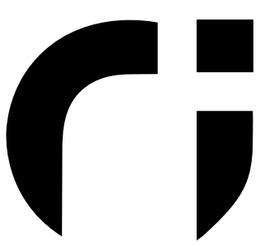
Location: QUINT.sol

Explanation:

The onlyOwner() modifier supports two accessibility flows depending on the state of the Developer_Access variable, however, the Developer_Access variable is always false, and no functionality allows it to change state. This results in code incorrectness in regards to the logical implementation of the documentation.

Recommendation:

Add a function that allows the owner to change the state of the Developer_Access variable, otherwise the developer access flows through onlyOwner() will not be reachable. It might be a good idea to change the name of the function as well if it is going to support multiple roles, to support clarity of understanding.



OBSN-03

Location: QUINT.sol

Explanation:

The remove_Random_Tokens(...) function allows the owner to withdraw tokens accumulated in the smart contract. This function can be chained with other functions such as the set_Limits_For_Swap_Trigger(...) function to create sophisticated exploits that can result in increased investor risk.

Recommendation:

Add a sanity check that prevents the owner from withdrawing QUINT tokens through the remove_Random_Tokens(...) function, and the QUINT token is not random in this particular environment.

OBSN-04

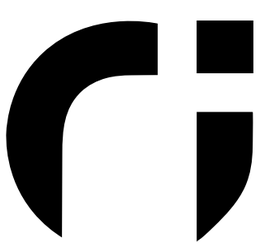
Location: QUINT.sol

Explanation:

The owner has the power to set a global maximum for transfer amounts to 1 by executing the set_Limits_For_Wallets(...) function, followed by executing the mapping_limitExempt(...) function on wallets of choice to give them the flexibility to sell arbitrary amounts of tokens. This creates a potential honeypot scenario placing all wallets subject to the transfer limitations into a slow-motion selling mode by only being able to sell 1 token at a time.

Recommendation:

Limitations on transfer amounts go against the nature of decentralization, if present these types of restrictions should preferably be guarded by time-based conditions and not be centralized and freely available.



OBSN-05

Location: QUINT.sol

Explanation:

The code implementation does not align with the documentation. The code comments in the _transfer(...) function define that sniping bots will be blocked for the first 20 blocks, however, the implementation only blocks sniping bots for the first block.

Recommendation:

The code should match the documentation to avoid behavioral inconsistencies. Sniping bots can still be used after the desired block number, so whether this approach is better than a i.e. a whitelist should be discussed. Change the code to be in alignment with the documentation.

OBSN-06

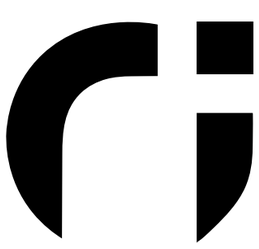
Location: QUINT.sol

Explanation:

The openTrade() function can be called unlimited times and on every function execution, the launchBlock variable will be updated. No functionality allows the trading of the token to be paused so this function should not be executable more than once.

Recommendation:

Add a sanity check in the openTrade() function that ensures that the function is only executable while the tradeOpen variable is false. This means the openTrade() function will only be executable once based on the current implementation.



OBSN-07

Location: QUINT.sol

Explanation:

The function signature type should be as explicit as possible. When a function is executed to read a value from storage rather than an input argument, then it should be marked with the view instead of pure keyword.

Recommendation:

The mutability of the following functions can be changed from pure to view since they only read from contract storage, and to align with industry-standard ERC20.sol implementations (see the latest OpenZeppelin repository on Github).

name()
decimals()
symbol()

OBSN-08

Location: QUINT.sol

Explanation:

The code does not follow a particular styling guide and this lowers the readability of the code and ultimately the quality as well. Also, some functions such as mapping_isPair(...) are misleading because it has a function parameter named wallet while the functionality is meant to be used for liquidity pool pairs which are never wallets.

Recommendation:

Use the same styling guide throughout the entire smart contract. The styling guide introduced in the Solidity Documentation is respected and used by most projects.

OBSN-09

Location: QUINT.sol

Explanation:

Looping over arrays of unknown size should be avoided when possible. In this particular scenario, looping over the _excluded array can cause an infinity loop in the _getCurrentSupply() function if the gas execution limit of the function exceeds the gas block limit due to a large array size.

Recommendation:

If looping over arrays of unknown size is unavoidable, then make sure to keep the array size small to avoid infinity loops. The Rewards_Exclude_Wallet(...) function should be used with good operational hygiene.

OBSN-10

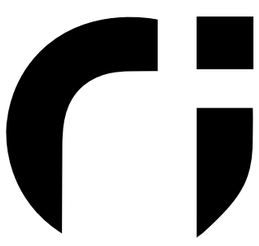
Location: QUINT.sol

Explanation:

Multiple functions in the code such as the openTrade() function use raw numbers in calculations. This should be avoided to increase the code legibility and maintainability.

Recommendation:

Replace magical numbers with contract variables whenever possible to avoid behavioral inconsistencies, improve readability, and to preserve legibility and maintainability.



OBSN-11

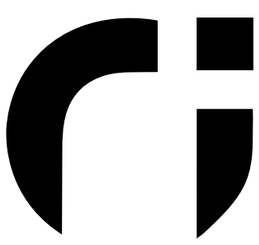
Location: QUINT.sol

Explanation:

Several functions can create sensitive state changes. The set_Fee_Distribution(...) function is sensitive to the tokenomics and positioning of an investor. Sensitive state changes like this should fire an event on-chain for improved transparency. It would be great to add events to the functions related to Multi-Sig activities as well.

Recommendation:

Create events for all functions that result in sensitive state changes and emit the events anytime the functions are executed to improve transparency and openness.



Some security issues were found during the assessment of the smart contract in scope and updates are required to pass our auditing process.

Tokenizing real-world assets and fusing the virtual world with the physical is one of the most important challenges of web3 adoption and the QUINT project aims to offer a solution consisting of a suite of products.

The smart contract in scope had some issues which need to be fixed by the team to ensure that the smart contract works as intended.

A. Classification List

TYPE	DESCRIPTION
Gas Optimization	Gas Optimization findings refer to code improvements that do not affect the functionality of the code but execute more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Mathematical Operations	Mathematical Operations details findings related to mishandling of mathematical formulas such as integer overflows, an incorrect order of operations, or percentage precisioning et al.
Logical Issue	Logical Issue findings defines faults in the logic of chained functions or particular expressions, such as an incorrect implementation of incentive designs, vesting schemes, and similar.
Control Flow	Control Flow findings concern the access control imposed on functions, such as overpowered access functions being executable by anyone under certain circumstances or control flow hijackings like reentrancy attacks.
Volatile Code	Volatile Code findings refer to code implementations that behave unexpectedly on particular edge cases that may result in exploitability or sensitive and unreliable code behavior.
Overpowered Design	Owerpowered Design findings describe code that entails a certain amount of trust in centralized entities such as an owner not behaving maliciously and to maintain code integrity.
Language Specific	Language Specific findings are issues that are related to the Solidity programming language such as incorrect usage of the delete keyword or conformity with language limitations.
Coding Style	Coding Style findings are primarily informational and they help to increase the quality of the codebase and easier maintainable by following a consistent styling guide.
Code Correctness	Code Correctness findings refer to functions that should seemingly behave similarly yet contain different code, legacy inheritance graph versioning, explicit visibility markings, etc.

A. Classification List

TYPE	DESCRIPTION
Magical Numbers	Magic Number findings refer to numerical values that are defined in the codebase in their raw format and should otherwise be specified as contract variables to increase code legibility and maintainability.
Compiler Error	Compiler Error findings refer to an issue in the implementation of a segment of the code that renders it impossible to compile using the specified version of the codebase.
Dead Code	Code that otherwise does not affect the functionality of the codebase and can be safely omitted to avoid code bloat and to increase the overall quality of the codebase.

B. Source Code Fingerprints

FILE	FINGERPRINT (SHA-256 Checksum)
QUINT.sol	cb755e35f06a554ec4c02e522700b76a4d3f6f5e78574c52783a8c5d82bad8c3