Security Assessment

lqt-core

May 15th, 2021

Summary

This report has been prepared for lqt-core smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Additionally, this audit is based on a premise that all external smart contracts are implemented safely.

The security assessment resulted in 5 findings that ranged from minor to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	lqt-core
Description	Liquidifty brings more crosschain liquidity to the NFT market.
Platform	BSC
Language	Solidity
Codebase	https://github.com/liquidifty/lqt-core/tree/fbc88a9f771992fc1f0d858e4566cd744c1d5cac
Commits	fbc88a9f771992fc1f0d858e4566cd744c1d5cac

Audit Summary

Delivery Date	May 15, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Total Issues	5
Critical	0
Major	0
Medium	0
Minor	1
Informational	4
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
LQT	LQT.sol	3f9e7c41697d5fe288fe1b3a1a035f671f47706eb137271452be0f3b761820cd
VES	Vesting.sol	3c31fd0914a2c6945d75ffd02484c91227ead9334d8b1f8b5e6aa1909276469f

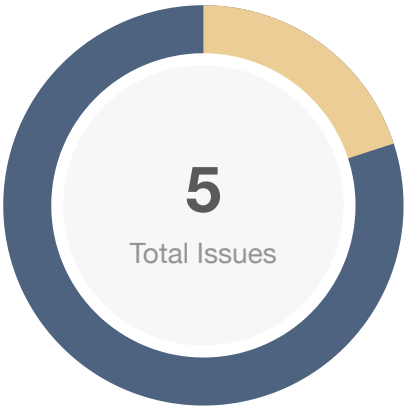
Centralization Roles

The LQT smart contract introduces a role.

Owner

- setBalances : distribute `lqt` tokens to holders, only used in `Vesting.sol`.

Findings



Critical	0 (0.00%)
Major	0 (0.00%)
Medium	0 (0.00%)
Minor	1 (20.00%)
Informational	4 (80.00%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
LQT-01	Missing Zero Address Validation	Logical Issue	Informational	Declined
LQT-02	Missing Emit Events	Coding Style	Informational	Partially Resolved
VES-01	Missing Emit Events	Coding Style	Informational	Partially Resolved
VES-02	Proper Usage of “public” And “external” Type	Gas Optimization	Informational	Resolved
VES-03	Check Effect Interaction Pattern Violated	Logical Issue	Minor	Resolved

LQT-01 | Missing Zero Address Validation

Category	Severity	Location	Status
Logical Issue	Informational	LQT.sol: 395	Declined

Description

Addresses should be checked before assignment to make sure they are not zero addresses.

Recommendation

Consider adding a zero check.

Alleviation

No alleviation.

LQT-02 | Missing Emit Events

Category	Severity	Location	Status
Coding Style	Informational	LQT.sol: 405, 429, 503	Partially Resolved

Description

Some functions should be able to emit events as notifications to customers because they change the status of sensitive variables or call important processes. This suggestion is not limited to these codes but also applies to other similar codes.

Recommendation

Consider adding an emit after changing the status of variables or calling important processes.

Alleviation

The team changed some code. Code change was applied in commit `0ac934bd99a50d17bb5c8e35aa1728f67946e1f3`.

VES-01 | Missing Emit Events

Category	Severity	Location	Status
Coding Style	Informational	Vesting.sol: 627	Partially Resolved

Description

Some functions should be able to emit events as notifications to customers because they change the status of sensitive variables or call important processes. This suggestion is not limited to these codes but also applies to other similar codes.

Recommendation

Consider adding an emit after changing the status of variables or calling important processes.

Alleviation

The team changed some code. Code change was applied in commit `0ac934bd99a50d17bb5c8e35aa1728f67946e1f3`.

VES-02 | Proper Usage of “public” And “external” Type

Category	Severity	Location	Status
Gas Optimization	Informational	Vesting.sol: 604, 608, 612, 654	Resolved

Description

`public` functions that are never called by the contract could be declared `external`.

Recommendation

Consider using the `external` attribute for functions never called from the contract.

Alleviation

The team heeded our advice and changed related code. Code change was applied in commit `0ac934bd99a50d17bb5c8e35aa1728f67946e1f3`.

VES-03 | Check Effect Interaction Pattern Violated

Category	Severity	Location	Status
Logical Issue	Minor	Vesting.sol: 654~662	Resolved

Description

The order of external call/transfer and storage manipulation must follow a check effect interaction pattern. This suggestion is not limited to these codes but also applies to other similar codes.

Recommendation

We advise the client to check if storage manipulation is before the external call/transfer operation by considering the following modification:

```
uint256 amount = availableAmount(msg.sender);
require(_balances[msg.sender] > 0 && amount > 0, "Nothing to release");
_balances[msg.sender] = _balances[msg.sender].sub(amount);
_released[msg.sender] = _released[msg.sender].add(amount);
_totalReleased = _totalReleased.add(amount);
lqtToken.safeTransfer(msg.sender, amount);
emit Release(msg.sender, amount);
```

Alleviation

The team heeded our advice and changed related code. Code change was applied in commit `0ac934bd99a50d17bb5c8e35aa1728f67946e1f3`.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.